

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

```
1  """A module to find the required attenuation."""
2  import decimal as dec
3
4
5  def attlist(self):
6      """Generate a list of all the values in the attenuation sheet."""
7      listatt = []
8      for row in self.dsa.iter_rows(row_offset=2):
9          if row[self.dsas2128].value is not None:
10             # Adds value to list if it actually has a value
11             listatt.append(row[self.dsas2128].value)
12     return listatt
13
14
15     def closest(self, attlist):
16         """Find closest attenuation to target."""
17         closest = min(attlist, key=lambda x: abs(
18             dec.Decimal(x) - dec.Decimal(self.targetatt)))
19         closest = dec.Decimal(closest)
20         return closest
21
22
23     def attenuationsearch(self, attlist, closest):
24         """Search for the most accurate attenuation."""
25         for row in self.dsa.iter_rows():
26             if row[self.dsas2128].value == closest:
27                 att2128 = closest.quantize(dec.Decimal(
28                     '.001'), rounding=dec.ROUND_HALF_UP)
29                 row28 = row[self.dsastate28].value
30                 att2124 = dec.Decimal(row[self.dsas2124].value).quantize(
31                     dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
32                 att2132 = dec.Decimal(row[self.dsas2132].value).quantize(
33                     dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
34                 phase2124 =
35                     ↪ dec.Decimal(row[self.dsaphase2124].value).quantize(
36                         dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
37                 phase2128 =
38                     ↪ dec.Decimal(row[self.dsaphase2128].value).quantize(
39                         dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
40                 phase2132 =
41                     ↪ dec.Decimal(row[self.dsaphase2132].value).quantize(
42                         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 =
14                 ↪ dec.Decimal(row[self.phase24].value).quantize(
15                     dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
16                 phasehigh =
17                 ↪ dec.Decimal(row[self.phase32].value).quantize(
18                     dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
19                 phasediff = phasehigh - phaselow
20
21             return {'phase1': self.targetphase, 'row1': row1, 'att1': att1,
22                 ↪      'phasediff': phasediff, 'source1': 's180', 'totalphase':
23                 ↪      att1, 'total': self.targetphase}
24     else:
25         closest = min(set180, key=lambda x: abs(

```

```

22         dec.Decimal(x) - dec.Decimal(self.targetphase)))
23     for row in self.s180.iter_rows():
24         if row[self.phase28].value == closest:
25             closestround = closest.quantize(
26                 dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
27             row1 = int(row[0].value)
28             att1 = dec.Decimal(row[self.att28].value).quantize(
29                 dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
30             phaselow =
31                 ↪ dec.Decimal(row[self.phase24].value).quantize(
32                     dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
33             phasehigh =
34                 ↪ dec.Decimal(row[self.phase32].value).quantize(
35                     dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
36             phasediff = phasehigh - phaselow
37
38     return {'phase1': closestround, 'row1': row1, 'att1': att1,
39             ↪ 'phasediff': phasediff, 'source1': 's180', 'totalphase':
40             ↪ att1, 'total': closestround}
41
42 def check90(self, set90):
43     """90 degrees of magic."""
44     dec.getcontext().prec = 6
45     if self.targetphase in set90:
46         for row in self.s90.iter_rows():
47             if row[self.phase28].value == self.targetphase:
48                 row1 = int(row[0].value)
49                 att1 = dec.Decimal(row[self.att28].value).quantize(
50                     dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
51                 phaselow =
52                     ↪ dec.Decimal(row[self.phase24].value).quantize(
53                         dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
54                 phasehigh =
55                     ↪ dec.Decimal(row[self.phase32].value).quantize(
56                         dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
57                 phasediff = phasehigh - phaselow
58
59         return {'phase1': self.targetphase, 'row1': row1, 'att1': att1,
60                 ↪ 'phasediff': phasediff, 'source1': 's90', 'totalphase':
61                 ↪ att1, 'total': self.targetphase}
62
63     else:
64         closest = min(set90, key=lambda x: abs(
65             dec.Decimal(x) - dec.Decimal(self.targetphase)))

```

```

59     for row in self.s90.iter_rows():
60         if row[self.phase28].value == closest:
61             closestround = closest.quantize(
62                 dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
63             row1 = int(row[0].value)
64             att1 = dec.Decimal(row[self.att28].value).quantize(
65                 dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
66             phaselow =
67                 ↳ dec.Decimal(row[self.phase24].value).quantize(
68                     dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
69             phasehigh =
70                 ↳ dec.Decimal(row[self.phase32].value).quantize(
71                     dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
72             phasediff = phasehigh - phaselow
73
74     return {'phase1': closestround, 'row1': row1, 'att1': att1,
75             ↳ 'phasediff': phasediff, 'source1': 's90', 'totalphase':
76             ↳ att1, 'total': closestround}
77
78 def check45(self, set45):
79     """45 Degrees of magic.
80
81     Parameters
82     -----
83     self, set45, set45
84     Returns
85     -----
86     phase1, row1, att1, phasediff
87     """
88     dec.getcontext().prec = 6
89     if self.targetphase in set45:
90         for row in self.s45.iter_rows():
91             if row[self.phase28].value == self.targetphase:
92                 row1 = int(row[0].value)
93                 att1 = dec.Decimal(row[self.att28].value).quantize(
94                     dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
95                 phaselow =
96                     ↳ dec.Decimal(row[self.phase24].value).quantize(
97                         dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
98                 phasehigh =
99                     ↳ dec.Decimal(row[self.phase32].value).quantize(
100                         dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
101                 phasediff = phasehigh - phaselow

```

```

98         return {'phase1': self.targetphase, 'row1': row1, 'att1': att1,
99                ↪ 'phasediff': phasediff, 'source1': 's45', 'totalphase':
100                ↪ att1, 'total': self.targetphase}
101
102     else:
103         closest = min(set45, key=lambda x: abs(
104             dec.Decimal(x) - dec.Decimal(self.targetphase)))
105         for row in self.s45.iter_rows():
106             if row[self.phase28].value == closest:
107                 closestround = closest.quantize(
108                     dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
109                 row1 = int(row[0].value)
110                 att1 = dec.Decimal(row[self.att28].value).quantize(
111                     dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
112                 phaselow =
113                 ↪ dec.Decimal(row[self.phase24].value).quantize(
114                     dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
115                 phasehigh =
116                 ↪ dec.Decimal(row[self.phase32].value).quantize(
117                     dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
118                 phasediff = phasehigh - phaselow
119
120         return {'phase1': closestround, 'row1': row1, 'att1': att1,
121                ↪ 'phasediff': phasediff, 'source1': 's45', 'totalphase':
122                ↪ att1, 'total': closestround}
123
124     def check225(self, set225):
125         """22.5 degrees of magic.
126
127         Parameters
128         -----
129         self, s225, set225
130         Returns
131         -----
132         phase1, row1, att1, phasediff
133         """
134         dec.getcontext().prec = 6
135         if self.targetphase in set225:
136             for row in self.s225.iter_rows():
137                 if row[self.phase28].value == self.targetphase:
138                     row1 = int(row[0].value)
139                     att1 = dec.Decimal(row[self.att28].value).quantize(
140                         dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
141                     phaselow =
142                     ↪ dec.Decimal(row[self.phase24].value).quantize(

```

```

137         dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
138     phasehigh =
139         ↪ dec.Decimal(row[self.phase32].value).quantize(
140         dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
141     phasediff = phasehigh - phaselow
142
143     return {'phase1': self.targetphase, 'row1': row1, 'att1': att1,
144             ↪ 'phasediff': phasediff, 'source1': 's225', 'totalphase':
145             ↪ att1, 'total': self.targetphase}
146
147 else:
148     closest = min(set225, key=lambda x: abs(
149         dec.Decimal(x) - dec.Decimal(self.targetphase)))
150     for row in self.s225.iter_rows():
151         if row[self.phase28].value == closest:
152             closestround = closest.quantize(
153                 dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
154             row1 = int(row[0].value)
155             att1 = dec.Decimal(row[self.att28].value).quantize(
156                 dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
157             phaselow =
158                 ↪ dec.Decimal(row[self.phase24].value).quantize(
159                 dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
160             phasehigh =
161                 ↪ dec.Decimal(row[self.phase32].value).quantize(
162                 dec.Decimal('.001'), rounding=dec.ROUND_HALF_UP)
163             phasediff = phasehigh - phaselow
164
165     return {'phase1': closestround, 'row1': row1, 'att1': att1,
166             ↪ 'phasediff': phasediff, 'source1': 's225', 'totalphase':
167             ↪ att1, 'total': closestround}
168
169 def checkall(self, set180, set90, set45, set225):
170     """Check to find most accurate solution so far.
171
172     Parameters
173     -----
174     self
175     Returns
176     -----
177     bestsol data dict
178     """
179     dec.getcontext().prec = 6
180     sol180 = check180(self, set180)
181     sol90 = check90(self, set90)

```

```

176     sol45 = check45(self, set45)
177     sol225 = check225(self, set225)
178     bestsollist = [sol180['phase1'], sol90[
179         'phase1'], sol45['phase1'], sol225['phase1']]
180
181     if sol180['phase1'] == self.targetphase:
182         return sol180
183
184     elif sol90['phase1'] == self.targetphase:
185         return sol90
186
187     elif sol45['phase1'] == self.targetphase:
188         return sol45
189
190     elif sol225['phase1'] == self.targetphase:
191         return sol225
192
193     else:
194         closest = min(bestsollist, key=lambda x: abs(
195             dec.Decimal(x) - dec.Decimal(self.targetphase)))
196         if closest == sol180['phase1']:
197             return sol180
198         elif closest == sol90['phase1']:
199             return sol90
200         elif closest == sol45['phase1']:
201             return sol45
202         elif closest == sol225['phase1']:
203             return sol225
204         else:
205             return None
206
207
208     def mostaccurate(self, bestresult, bestresult2, bestresult3,
209         ↪ bestresult4, sollist):
210         """Find most accurate solution and return."""
211         bestsol = min(sollist, key=lambda x: abs(
212             dec.Decimal(x) - dec.Decimal(self.targetphase)))
213         if bestsol == bestresult['total']:
214             return bestresult
215         elif bestsol == bestresult2['total']:
216             return bestresult2
217         elif bestsol == bestresult3['total']:
218             return bestresult3
219         elif bestsol == bestresult4['total']:
220             return bestresult4
221         else:

```


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

```

1  """Test all four phases in one."""
2  import decimal as dec
3  import os
4
5
6  def closest_finder(self):
7      """Find the closest value to a given value."""
8      combined = set(map(str.rstrip, open(
9          (os.path.join(os.path.dirname(__file__), '1809045225.txt')))))
10     closest = min(combined, key=lambda x: abs(
11         dec.Decimal(x) - dec.Decimal(self.targetphase)))
12     closest = dec.Decimal(closest)
13     return closest
14     del combined
15
16
17 def check(self, set180, set90, set45, set225, closest):
18     """Check to find the most accurate four phase solution."""
19     dec.getcontext().prec = 6
20     for i in set180:
21         for j in set90:
22             for k in set45:
23                 for l in set225:
24                     total = i + j + k + l
25                     if total == closest:
26                         for row in self.s180:
27                             if row[self.phase28].value == i:
28                                 row1 = int(row[0].value)
29                                 att1 =
                                     ↪ dec.Decimal(row[self.att28].value).quantize(

```

```

30         dec.Decimal('.001'),
           ↳ rounding=dec.ROUND_HALF_UP)
31 phaselow1 =
           ↳ dec.Decimal(row[self.phase24].value).quantize(
32         dec.Decimal('.001'),
           ↳ rounding=dec.ROUND_HALF_UP)
33 phasehigh1 =
           ↳ dec.Decimal(row[self.phase32].value).quantize(
34         dec.Decimal('.001'),
           ↳ rounding=dec.ROUND_HALF_UP)
35 phasediff1 = phasehigh1 - phaselow1
36
37 for row in self.s90:
38     if row[self.phase28].value == j:
39         row2 = int(row[0].value)
40         att2 =
           ↳ dec.Decimal(row[self.att28].value).quantize(
41         dec.Decimal('.001'),
           ↳ rounding=dec.ROUND_HALF_UP)
42 phaselow2 =
           ↳ dec.Decimal(row[self.phase24].value).quantize(
43         dec.Decimal('.001'),
           ↳ rounding=dec.ROUND_HALF_UP)
44 phasehigh2 =
           ↳ dec.Decimal(row[self.att32].value).quantize(
45         dec.Decimal('.001'),
           ↳ rounding=dec.ROUND_HALF_UP)
46 phasediff2 = phasehigh2 - phaselow2
47
48 for row in self.s45:
49     if row[self.phase28].value == k:
50         row3 = int(row[0].value)
51         att3 =
           ↳ dec.Decimal(row[self.att28].value).quantize(
52         dec.Decimal('.001'),
           ↳ rounding=dec.ROUND_HALF_UP)
53 phaselow3 =
           ↳ dec.Decimal(row[self.att24].value).quantize(
54         dec.Decimal('.001'),
           ↳ rounding=dec.ROUND_HALF_UP)
55 phasehigh3 =
           ↳ dec.Decimal(row[self.att32].value).quantize(
56         dec.Decimal('.001'),
           ↳ rounding=dec.ROUND_HALF_UP)

```

```

57         phasediff3 = phasehigh3 - phaselow3
58
59     for row in self.s225:
60         if row[self.phase28].value == 1:
61             row4 = int(row[0].value)
62             att4 =
63                 ↳ dec.Decimal(row[self.att28].value).quantize(
64                     dec.Decimal('.001'),
65                     ↳ rounding=dec.ROUND_HALF_UP)
66             phaselow4 =
67                 ↳ dec.Decimal(row[self.phase24].value).quantize(
68                     dec.Decimal('.001'),
69                     ↳ rounding=dec.ROUND_HALF_UP)
70             phasehigh4 =
71                 ↳ dec.Decimal(row[self.phase32].value).quantize(
72                     dec.Decimal('.001'),
73                     ↳ rounding=dec.ROUND_HALF_UP)
74             phasediff4 = phasehigh4 - phaselow4
75
76     totalatt = att1 + att2 + att3 + att4
77     return {'phase1':
78         ↳ i.quantize(dec.Decimal('.001'),
79             ↳ rounding=dec.ROUND_HALF_UP), 'row1': row1,
80         ↳ 'att1': att1, 'phasediff1': phasediff1,
81         ↳ 'source1': 's180', 'phase2':
82         ↳ j.quantize(dec.Decimal('.001'),
83             ↳ rounding=dec.ROUND_HALF_UP), 'row2': row2,
84         ↳ 'att2': att2, 'phasediff2': phasediff2,
85         ↳ 'source2': 's90', 'phase3':
86         ↳ k.quantize(dec.Decimal('.001'),
87             ↳ rounding=dec.ROUND_HALF_UP), 'row3': row3,
88         ↳ 'att3': att3, 'phasediff3': phasediff3,
89         ↳ 'source3': 's45', 'phase4':
90         ↳ l.quantize(dec.Decimal('.001'),
91             ↳ rounding=dec.ROUND_HALF_UP), 'row4': row4,
92         ↳ 'att4': att4, 'phasediff4': phasediff4,
93         ↳ 'source4': 's225', 'total':
94         ↳ total.quantize(dec.Decimal('.001'),
95             ↳ rounding=dec.ROUND_HALF_UP), 'totalatt':
96         ↳ totalatt.quantize(dec.Decimal('.001'),
97             ↳ 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 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

```
1  """To generate a look-up table."""
2  import decimal as dec
3  from src import extras as ex
4  from src import TwoPhase as twp
5  from src import ThreePhase as thp
6  from src import FourPhase as fp
7  import tabulate
8  import os
9  import csv
10
11
12  def tablegen(self):
13      """Generate look up table for phase."""
14      list180 = []
15      for row in self.s180.iter_rows(row_offset=2):
16          if row[self.phase28].value is not None:
17              list180.append(dec.Decimal(row[self.phase28].value))
18      set180 = set(list180)
19
20      list90 = []
21      for row in self.s90.iter_rows(row_offset=2):
22          if row[self.phase28].value is not None:
23              list90.append(dec.Decimal(row[self.phase28].value))
24      set90 = set(list90)
25
26      list45 = []
27      for row in self.s45.iter_rows(row_offset=2):
28          if row[self.phase28].value is not None:
29              list45.append(dec.Decimal(row[self.phase28].value))
```

```

30 set45 = set(list45)
31
32 list225 = []
33 for row in self.s225.iter_rows(row_offset=2):
34     if row[self.phase28].value is not None:
35         list225.append(dec.Decimal(row[self.phase28].value))
36 set225 = set(list225)
37 headers = ["Target value", "180 Used", "90 Used", "45 Used",
38            "22.5 Used", "180 Setting", "90 Setting", "45 Setting",
39            ↪ "22.5 Setting"]
40 table = []
41 for i in range(1, 65):
42     self.targetphase = (-360 / 64) * i
43     bestresult = ex.checkall(self, set180, set90, set45, set225)
44     bestresult2 = twp.checkall(self, set180, set90, set45, set225)
45     bestresult3 = thp.checkall(self, set180, set90, set45, set225)
46     closest = fp.closest_finder(self)
47     bestresult4 = fp.check(self, set180, set90, set45, set225,
48        ↪ closest)
49     sollist = [bestresult['total'], bestresult2['total'],
50        bestresult3['total'], bestresult4['total']]
51     bestphase = ex.mostaccurate(
52         self, bestresult, bestresult2, bestresult3, bestresult4,
53         ↪ sollist)
54     s180present = 0
55     s90present = 0
56     s45present = 0
57     s225present = 0
58     if 'source1' in bestphase:
59         if bestphase['source1'] == 's180':
60             s180present = 1
61         elif bestphase['source1'] == 's90':
62             s90present = 1
63         elif bestphase['source1'] == 's45':
64             s45present = 1
65         elif bestphase['source1'] == 's225':
66             s225present = 1
67
68     if 'source2' in bestphase:
69         if bestphase['source2'] == 's90':
70             s90present = 2
71         elif bestphase['source2'] == 's45':
72             s45present = 2
73         elif bestphase['source2'] == 's225':
74             s225present = 2

```

```

73     if 'source3' in bestphase:
74         if bestphase['source3'] == 's45':
75             s45present = 3
76         elif bestphase['source3'] == 's225':
77             s225present = 3
78
79     if 'source4' in bestphase:
80         if bestphase['source4'] == 's225':
81             s225present = 4
82     if s180present == 1:
83         s180setting = '{0:02b}'.format(bestphase['row1'])
84     else:
85         s180setting = '{0:02b}'.format(2)
86
87     if s90present == 1:
88         s90setting = '{0:06b}'.format(bestphase['row1'])
89     elif s90present == 2:
90         s90setting = '{0:06b}'.format(bestphase['row2'])
91         s90present = 1
92     else:
93         s90setting = '{0:06b}'.format(32)
94
95     if s45present == 1:
96         s45setting = '{0:09b}'.format(bestphase['row1'])
97     elif s45present == 2:
98         s45setting = '{0:09b}'.format(bestphase['row2'])
99         s45present = 1
100    elif s45present == 3:
101        s45setting = '{0:09b}'.format(bestphase['row3'])
102        s45present = 1
103    else:
104        s45setting = '{0:09b}'.format(256)
105
106    if s225present == 1:
107        s225setting = '{0:09b}'.format(bestphase['row1'])
108    elif s225present == 2:
109        s225setting = '{0:09b}'.format(bestphase['row2'])
110        s225present = 1
111    elif s225present == 3:
112        s225setting = '{0:09b}'.format(bestphase['row3'])
113        s225present = 1
114    elif s225present == 4:
115        s225setting = '{0:09b}'.format(bestphase['row4'])
116        s225present = 1
117    else:
118        s225setting = '{0:09b}'.format(256)

```

```

119
120         endlist = [self.targetphase, s180present, s90present,
121                     ↪ s45present,
122                     s225present, s180setting, s90setting, s45setting,
123                     ↪ s225setting]
124         table.append(endlist)
125
126     print(tabulate.tabulate(table, headers, tablefmt="fancy_grid"))
127     with open(os.path.join(
128         os.path.dirname(__file__), 'phasetable.csv'), 'w') as
129         ↪ csvfile:
130         writer = csv.writer(csvfile)
131         [writer.writerow(r) for r in table]
132     return
133
134 def atttablegen(self):
135     """Generate a table for attenuation."""
136     listatt = []
137     for row in self.dsa.iter_rows(row_offset=2):
138         if row[self.dsas2128].value is not None:
139             listatt.append(dec.Decimal(row[self.dsas2128].value))
140     setatt = set(listatt)
141     headers = ["Target", "Attenuation", "Setting"]
142     table = []
143     for i in range(1, 65):
144         self.targetatt = dec.Decimal((-24 / 64) * i)
145         if self.targetatt in setatt:
146             for row in self.dsa.iter_rows():
147                 if row[self.dsas2128].value == self.targetatt:
148                     row28 = row[self.dsas2128].value
149                     att2128 = self.targetatt.quantize(
150                         dec.Decimal('.001'),
151                         ↪ rounding=dec.ROUND_HALF_UP)
152                     att2124 =
153                         ↪ dec.Decimal(row[self.dsas2124].value).quantize(
154                             dec.Decimal('.001'),
155                             ↪ rounding=dec.ROUND_HALF_UP)
156                     att2132 =
157                         ↪ dec.Decimal(row[self.dsas2132].value).quantize(
158                             dec.Decimal('.001'),
159                             ↪ rounding=dec.ROUND_HALF_UP)
160
161         else:
162             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 =
164                 ↪ dec.Decimal(row[self.dsas2124].value).quantize(
165                     dec.Decimal('.001'),
166                     ↪ rounding=dec.ROUND_HALF_UP)
167             att2132 =
168                 ↪ dec.Decimal(row[self.dsas2132].value).quantize(
169                     dec.Decimal('.001'),
170                     ↪ rounding=dec.ROUND_HALF_UP)
171
172     DSAssetting = '{0:012b}'.format(row28)
173     table.append([self.targetatt, att2128, DSAssetting])
174
175     print(tabulate.tabulate(table, headers, tablefmt="fancy_grid"))
176     with open(os.path.join(
177         os.path.dirname(__file__), 'atttable.csv'), 'w') as csvfile:
178         writer = csv.writer(csvfile)
179         [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

```



```

4  from src import AttenuationSearch as ats
5
6
7  def minattvar(self):
8      """Search for minimum amplitude variation across frequency."""
9      listatt = []
10     totallist = []
11     attvar = []
12     # Iterates through all the rows in the spreadsheet from the third
13     ↪ row
14     # onwards
15     for row in self.dsa.iter_rows(row_offset=2):
16         if row[self.dsas2128].value is not None:
17             # Adds the value to a list if it actually has a value, and
18             ↪ isn't
19             # blank
20             listatt.append(row[self.dsas2128].value)
21     closest_values = nsmaallest(int(self.k), listatt, key=lambda x: abs(
22     dec.Decimal(x) - dec.Decimal(self.targetatt))) # Finds the k
23     ↪ closest numbers to a given value from the list
24     # Converts the values in closest_values to a decimal from a float.
25     ↪ This
26     # makes it easier to work with as it adds in rounding
27     closest = [dec.Decimal(i) for i in closest_values]
28     for i in closest:
29         # Calls the AttenuationSearch module to search for the various
30         ↪ values.
31         # It does this for each item in the closest list
32         resultsdict = ats.attenuationsearch(self, listatt, i)
33         variation = resultsdict['att2132'] - resultsdict['att2124']
34         # Adds a new value to the dictionary, with the key variation
35         resultsdict['variation'] = variation
36         totallist.append(resultsdict)
37         attvar.append(variation)
38     # Finds the smallest number in the list
39     minattdiff = min(abs(i) for i in attvar)
40     for m in totallist:
41         if abs(m['variation']) == minattdiff:
42             # If the value in the dictionary/key combo m['variation']
43             ↪ matches
44             # the minimum value, it gets returned and printed
45             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

```
1  """Find the minimum insertion loss, i.e., the minimum attenuation for a
   ↳ given phase."""
2  import decimal as dec
3  import os
4  from heapq import nsmallest
5  from src import FourPhase as fp
6
7
8  def mininsertloss(self, set180, set90, set45, set225):
9      """Find minimum insertion loss for a phase."""
10     dec.getcontext().prec = 6
11     combined = set(map(dec.Decimal, open((os.path.join(
12         os.path.dirname(__file__), '1809045225.txt')))))
13     closest_values = nsmallest(int(self.k), combined, key=lambda x:
   ↳ abs(
14         x - dec.Decimal(self.targetphase)))
15     closest = [dec.Decimal(i) for i in closest_values]
16     total_values = []
17     insertlosslist = []
18     del combined
19     for item in closest:
20         resultsdict = fp.check(self, set180, set90, set45, set225,
   ↳ item)
21         insertlosslist.append(resultsdict['totalatt'])
22         total_values.append(resultsdict)
23     mininsertloss = min(abs(i) for i in insertlosslist)
24     for m in total_values:
25         if abs(m['totalatt']) == mininsertloss:
26             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

```
1  """A module to find the minimum variation in phase across frequency"""
2  import decimal as dec
3  from heapq import nsmallest
4  from src import AttenuationSearch as ats
5
```

```

6
7 def minampvar(self):
8     """Search for minimum variation in phase across attenuation
9     ↪ frequency."""
10    listatt = []
11    totallist = []
12    ampvar = []
13    for row in self.dsa.iter_rows(row_offset=2):
14        if row[self.dsas2128].value is not None:
15            listatt.append(row[self.dsas2128].value)
16    closest_values = nsmallest(int(self.k), listatt, key=lambda x: abs(
17        dec.Decimal(x) - dec.Decimal(self.targetatt)))
18    closest = [dec.Decimal(i) for i in closest_values]
19    for i in closest:
20        resultsdict = ats.attenuationsearch(self, listatt, i)
21        # Finds total variation in phase
22        variation = resultsdict['phase2132'] - resultsdict['phase2124']
23        resultsdict['variation'] = variation
24        totallist.append(resultsdict)
25        ampvar.append(variation)
26    minampdiff = min(abs(i) for i in ampvar)
27    for m in totallist:
28        if abs(m['variation']) == minampdiff:
29            return m # Returns the dictionary with the minimum phase
30            ↪ 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
10     ↪ frequency."""
11      dec.getcontext().prec = 6
12      combined = set(map(dec.Decimal, open(

```

```

12         (os.path.join(os.path.dirname(__file__), '1809045225.txt')))))
        ↳ # Gets the path to the directory that the program is
        ↳ running in
13     closest_values = nsmaallest(int(self.k), combined, key=lambda x:
        ↳ abs(
14         x - dec.Decimal(self.targetphase)))
15     closest = [dec.Decimal(i) for i in closest_values]
16     total_values = []
17     phasedifflist = []
18     del combined
19     for item in closest:
20         resultsdict = fp.check(self, set180, set90, set45, set225,
        ↳ item)
21         totalphasediff = resultsdict['phasediff1'] + resultsdict[
22             'phasediff2'] + resultsdict['phasediff3'] +
        ↳ resultsdict['phasediff4']
23         resultsdict['totalphasediff'] = totalphasediff
24         phasedifflist.append(totalphasediff)
25         total_values.append(resultsdict)
26     minphasediff = min(phasedifflist)
27     for m in total_values:
28         if m['totalphasediff'] == minphasediff:
29         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

```

1     """Module to work on more ideas for MPAC tuning."""
2     import os
3     import decimal as dec
4     import openpyxl as xl
5     from src import extras as ex
6     from src import TwoPhase as tp
7     from src import ThreePhase as thp
8     from src import FourPhase as fp
9     from src import AttenuationSearch as ats
10    from src import lookuptablegenerator as lutg
11    from src import minphasevariation as mpv
12    from src import mininsertloss as mil
13    from src import minphaseatt as mpa
14    from src import minattvar as mav
15    from src import numberformatting as nf
16

```

```

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

```

```

60     print("D - the value with least variation across
        ↳ frequency")
61     print("I - the value with minimum insertion loss")
62     print("A - the attenuation value with minimum variation
        ↳ across frequency")
63     print(
64         "P - the attenuation value with minimum phase
        ↳ difference across frequency")
65     minvarchoice = input()
66
67     list180 = []
68     for row in self.s180.iter_rows(row_offset=2):
69         if row[self.phase28].value is not None:
70
71             ↳ list180.append(dec.Decimal(row[self.phase28].value))
72     set180 = set(list180)
73
74     list90 = []
75     for row in self.s90.iter_rows(row_offset=2):
76         if row[self.phase28].value is not None:
77             list90.append(dec.Decimal(row[self.phase28].value))
78     set90 = set(list90)
79
80     list45 = []
81     for row in self.s45.iter_rows(row_offset=2):
82         if row[self.phase28].value is not None:
83             list45.append(dec.Decimal(row[self.phase28].value))
84     set45 = set(list45)
85
86     list225 = []
87     for row in self.s225.iter_rows(row_offset=2):
88         if row[self.phase28].value is not None:
89
90             ↳ list225.append(dec.Decimal(row[self.phase28].value))
91     set225 = set(list225)
92     if minvarchoice == "D":
93         self.targetphase = input(
94             "Please enter the desired phase shift for 28GHz")
95         self.k = input("How many values would you like to
96             ↳ search?")
97         minphase = mpv.minvariation(self, set180, set90, set45,
98             ↳ set225)
99         print("The closest value is " + str(minphase[
100             'total']]) + ", giving a total variation of " +
101             ↳ 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,
109            ↪ set225)
110        bestresult2 = tp.checkall(self, set180, set90, set45,
111            ↪ set225)
112        bestresult3 = thp.checkall(self, set180, set90, set45,
113            ↪ set225)
114        closest = fp.closest_finder(self)
115        bestresult4 = fp.check(
116            self, set180, set90, set45, set225, closest)
117        sollist = [bestresult['total'], bestresult2['total'],
118            bestresult3['total'], bestresult4['total']]
119        bestphase = ex.mostaccurate(
120            self, bestresult, bestresult2, bestresult3,
121            ↪ bestresult4, sollist)
122        attlist = ats.attlist(self)
123        closest = ats.closest(self, attlist)
124        bestatt = ats.attenuationsearch(self, attlist, closest)
125        print("The best result is " + str(bestphase[
126            'total'])) + " and the best attenuation is " +
127            ↪ str(bestatt['att2128']))
128        formatted = nf.phaseformat(self, bestphase)
129        formatatt = nf.attformat(self, bestatt)
130        print("The settings you need for this are: ")
131        print("180: " + formatted['s180setting'])
132        print("90: " + formatted['s90setting'])
133        print("45: " + formatted['s45setting'])
134        print("22.5: " + formatted['s225setting'])
135        print("Attenuation: " + formatatt)
136    elif minvarchoice == "I":
137        self.targetphase = input(
138            "Please enter the desired phase shift for 28GHz")
139        self.k = input("How many values would you like to
140            ↪ search?")
141        mininsert = mil.mininsertloss(

```

```

136         self, set180, set90, set45, set225)
137     print("The best value is " + str(mininsert[
138         'total']) + ", giving a total insertion loss of "
        ↪ + str(mininsert['totalatt']))
139     formatted = nf.phaseformat(self, mininsert)
140     print("The settings you need for this are: ")
141     print("180: " + formatted['s180setting'])
142     print("90: " + formatted['s90setting'])
143     print("45: " + formatted['s45setting'])
144     print("22.5: " + formatted['s225setting'])
145     elif minvarchoice == "P":
146         self.targetatt = input(
147             "Please enter the desired attenuation for 28GHz")
148         self.k = input("How many values would you like to
        ↪ search?")
149         minamp = mpa.minampvar(self)
150         print("The best value is " + str(minamp['att2128']) +
151             ", giving a variation of " +
        ↪ str(minamp['variation']))
152         formatted = nf.attformat(self, minamp)
153         print("The setting you need is: " + formatted)
154     elif minvarchoice == "A":
155         self.targetatt = input(
156             "Please enter the desired attenuation for 28GHz")
157         self.k = input("How many values would you like to
        ↪ search?")
158         minatt = mav.minattvar(self)
159         print("The best attenuation is " +
160             str(minatt['att2128']) + ", giving a variation of
        ↪ " + str(minatt['variation']))
161         resultsetting = nf.attformat(self, minatt)
162         print("The setting you need is: " + resultsetting)
163
164     else:
165         print("Not a valid option")
166     elif consent == 'T':
167         option = input("Generate a table for (A)ttenuation or
        ↪ (P)hase?")
168         if option == "A":
169             lutg.atttablegen(self)
170         elif option == "P":
171             lutg.tablegen(self)
172         else:
173             print("That is not an option")
174

```



```

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 =

```

```

23         dec.Decimal('.001'),
           ↳ rounding=dec.ROUND_HALF_UP)
24 phaselow1 =
           ↳ dec.Decimal(row[self.phase24].value).quantize(
25         dec.Decimal('.001'),
           ↳ rounding=dec.ROUND_HALF_UP)
26 phasehigh1 =
           ↳ dec.Decimal(row[self.phase32].value).quantize(
27         dec.Decimal('.001'),
           ↳ rounding=dec.ROUND_HALF_UP)
28 phasediff1 = phasehigh1 - phaselow1
29
30 for row in self.s90:
31     if row[self.phase28].value == j:
32         row2 = int(row[0].value)
33         att2 =
           ↳ dec.Decimal(row[self.att28].value).quantize(
34         dec.Decimal('.001'),
           ↳ rounding=dec.ROUND_HALF_UP)
35 phaselow2 =
           ↳ dec.Decimal(row[self.phase24].value).quantize(
36         dec.Decimal('.001'),
           ↳ rounding=dec.ROUND_HALF_UP)
37 phasehigh2 =
           ↳ dec.Decimal(row[self.phase32].value).quantize(
38         dec.Decimal('.001'),
           ↳ rounding=dec.ROUND_HALF_UP)
39 phasediff2 = phasehigh2 - phaselow2
40
41 for row in self.s45:
42     if row[self.phase28].value == k:
43         row3 = int(row[0].value)
44         att3 =
           ↳ dec.Decimal(row[self.att28].value).quantize(
45         dec.Decimal('.001'),
           ↳ rounding=dec.ROUND_HALF_UP)
46 phaselow3 =
           ↳ dec.Decimal(row[self.att24].value).quantize(
47         dec.Decimal('.001'),
           ↳ rounding=dec.ROUND_HALF_UP)
48 phasehigh3 =
           ↳ dec.Decimal(row[self.att32].value).quantize(
49         dec.Decimal('.001'),
           ↳ rounding=dec.ROUND_HALF_UP)

```

```

50         phasediff3 = phasehigh3 - phaselow3
51
52     totalatt = att1 + att2 + att3
53     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)}
54
55
56     def check2(self, set180, set90, set225):
57         """Check 180, 90 and 22.5 arrays for solution."""
58         dec.getcontext().prec = 6
59         combined = set(map(str.rstrip, open(
60             (os.path.join(os.path.dirname(__file__), '18090225.txt')))))
61         closest = min(combined, key=lambda x: abs(
62             dec.Decimal(x) - dec.Decimal(self.targetphase)))
63         closest = dec.Decimal(closest)
64         for i in set180:
65             for j in set90:
66                 for k in set225:
67                     total = i + j + k
68                     if total == closest:
69                         for row in self.s180:
70                             if row[self.phase28].value == i:
71                                 row1 = int(row[0].value)
72                                 att1 =
73                                     ↪ dec.Decimal(row[self.att28].value).quantize(
74                                     ↪ dec.Decimal('.001'),
75                                     ↪ rounding=dec.ROUND_HALF_UP)
76                                 phaselow1 =
77                                     ↪ dec.Decimal(row[self.phase24].value).quantize(
78                                     ↪ dec.Decimal('.001'),
79                                     ↪ rounding=dec.ROUND_HALF_UP)

```

```

76         phasehigh1 =
77             ↪ dec.Decimal(row[self.phase32].value).quantize(
78                 dec.Decimal('.001'),
79                 ↪ rounding=dec.ROUND_HALF_UP)
80         phasediff1 = phasehigh1 - phaselow1
81
82     for row in self.s90:
83         if row[self.phase28].value == j:
84             row2 = int(row[0].value)
85             att2 =
86                 ↪ dec.Decimal(row[self.att28].value).quantize(
87                     dec.Decimal('.001'),
88                     ↪ rounding=dec.ROUND_HALF_UP)
89             phaselow2 =
90                 ↪ dec.Decimal(row[self.phase24].value).quantize(
91                     dec.Decimal('.001'),
92                     ↪ rounding=dec.ROUND_HALF_UP)
93             phasehigh2 =
94                 ↪ dec.Decimal(row[self.phase32].value).quantize(
95                     dec.Decimal('.001'),
96                     ↪ rounding=dec.ROUND_HALF_UP)
97             phasediff2 = phasehigh2 - phaselow2
98
99     for row in self.s225:
100         if row[self.phase28].value == k:
101             row3 = int(row[0].value)
102             att3 =

```

103

```

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)}

```

104

105

106 **def** check3(self, set180, set45, set225):107 **"""**Check 180, 45 and 22.5 arrays for solution.**"""**

108 dec.getcontext().prec = 6

109 combined = set(map(str.rstrip, open(
 ↪ (os.path.join(os.path.dirname(__file__), '18045225.txt')))))

110 closest = min(combined, key=lambda x: abs(
 ↪ dec.Decimal(x) - dec.Decimal(self.targetphase)))

111 closest = dec.Decimal(closest)

112 **for** i **in** set180:113 **for** j **in** set45:114 **for** k **in** set225:

115 total = i + j + k

116 **if** total == closest:117 **for** row **in** self.s180:118 **if** row[self.phase28].value == i:

119 row1 = int(row[0].value)

120 att1 =

121 ↪ dec.Decimal(row[self.att28].value).quantize(
 ↪ dec.Decimal('.001'),

122 ↪ rounding=dec.ROUND_HALF_UP)

123 phaselow1 =

124 ↪ dec.Decimal(row[self.phase24].value).quantize(
 ↪ dec.Decimal('.001'),

125 ↪ rounding=dec.ROUND_HALF_UP)

126 phasehigh1 =

↪ dec.Decimal(row[self.phase32].value).
 ↪ quantize(

```

127         dec.Decimal('.001'),
128         ↳ rounding=dec.ROUND_HALF_UP)
129     phasediff1 = phasehigh1 - phaselow1
130
131     for row in self.s45:
132         if row[self.phase28].value == j:
133             row2 = int(row[0].value)
134             att2 =
135                 ↳ dec.Decimal(row[self.att28].value).quantize(
136                 dec.Decimal('.001'),
137                 ↳ rounding=dec.ROUND_HALF_UP)
138             phaselow2 =
139                 ↳ dec.Decimal(row[self.att24].value).quantize(
140                 dec.Decimal('.001'),
141                 ↳ rounding=dec.ROUND_HALF_UP)
142             phasehigh2 =
143                 ↳ dec.Decimal(row[self.att32].value).quantize(
144                 dec.Decimal('.001'),
145                 ↳ rounding=dec.ROUND_HALF_UP)
146             phasediff2 = phasehigh2 - phaselow2
147
148     for row in self.s225:
149         if row[self.phase28].value == k:
150             row3 = int(row[0].value)
151             att3 =
152                 ↳ dec.Decimal(row[self.att28].value).quantize(
153                 dec.Decimal('.001'),
154                 ↳ rounding=dec.ROUND_HALF_UP)
155             phaselow3 =
156                 ↳ dec.Decimal(row[self.phase24].value).quantize(
157                 dec.Decimal('.001'),
158                 ↳ rounding=dec.ROUND_HALF_UP)
159             phasehigh3 =
160                 ↳ dec.Decimal(row[self.phase32].value).quantize(
161                 dec.Decimal('.001'),
162                 ↳ rounding=dec.ROUND_HALF_UP)
163             phasediff3 = phasehigh3 - phaselow3
164
165     totalatt = att1 + att2 + att3

```

153

```

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)}

```

154

155

156 **def** check4(self, set90, set45, set225):157 **"""**Check 90, 45 and 22.5 arrays for solution.**"""**

158 dec.getcontext().prec = 6

159 combined = set(map(str.rstrip, open(
160 (os.path.join(os.path.dirname(__file__), '9045225.txt')))))

161 closest = min(combined, key=lambda x: abs(
162 dec.Decimal(x) - dec.Decimal(self.targetphase)))

163 closest = dec.Decimal(closest)

164 **for** i **in** set90:165 **for** j **in** set45:166 **for** k **in** set225:

167 total = i + j + k

168 **if** total == closest:169 **for** row **in** self.s90:170 **if** row[self.phase28].value == i:

171 row1 = int(row[0].value)

172 att1 =

```

                            ↪ dec.Decimal(row[self.att28].value).quantize(
                            ↪ dec.Decimal('.001'),

```

173 ↪ rounding=dec.ROUND_HALF_UP)

174 phaselow1 =

```

                            ↪ dec.Decimal(row[self.phase24].value).quantize(
                            ↪ dec.Decimal('.001'),

```

175 ↪ rounding=dec.ROUND_HALF_UP)

176 phasehigh1 =

```

                            ↪ dec.Decimal(row[self.att32].value).quantize(

```

```

177         dec.Decimal('.001'),
178         ↳ rounding=dec.ROUND_HALF_UP)
179     phasediff1 = phasehigh1 - phaselow1
180
181     for row in self.s45:
182         if row[self.phase28].value == j:
183             row2 = int(row[0].value)
184             att2 =
185                 ↳ dec.Decimal(row[self.att28].value).quantize(
186                 dec.Decimal('.001'),
187                 ↳ rounding=dec.ROUND_HALF_UP)
188             phaselow2 =
189                 ↳ dec.Decimal(row[self.att24].value).quantize(
190                 dec.Decimal('.001'),
191                 ↳ rounding=dec.ROUND_HALF_UP)
192             phasehigh2 =
193                 ↳ dec.Decimal(row[self.att32].value).quantize(
194                 dec.Decimal('.001'),
195                 ↳ rounding=dec.ROUND_HALF_UP)
196             phasediff2 = phasehigh2 - phaselow2
197
198     for row in self.s225:
199         if row[self.phase28].value == k:
200             row3 = int(row[0].value)
201             att3 =
202                 ↳ dec.Decimal(row[self.att28].value).quantize(
203                 dec.Decimal('.001'),
204                 ↳ rounding=dec.ROUND_HALF_UP)
205             phaselow3 =
206                 ↳ dec.Decimal(row[self.phase24].value).quantize(
207                 dec.Decimal('.001'),
208                 ↳ rounding=dec.ROUND_HALF_UP)
209             phasehigh3 =
210                 ↳ dec.Decimal(row[self.phase32].value).quantize(
211                 dec.Decimal('.001'),
212                 ↳ rounding=dec.ROUND_HALF_UP)
213             phasediff3 = phasehigh3 - phaselow3
214
215     totalatt = att1 + att2 + att3

```


203

```

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)}

```

204

205

206 **def** checkall(self, set180, set90, set45, set225):

207 """Check for best three phase solution."""

208 dec.getcontext().prec = 6

209 sol1 = check1(self, set180, set90, set45)

210 sol2 = check2(self, set180, set90, set225)

211 sol3 = check3(self, set180, set45, set225)

212 sol4 = check4(self, set90, set45, set225)

213

214 sollist = [sol1['total'], sol2['total'], sol3['total'],

```

    ↪ sol4['total']]

```

215 **if** sol1['total'] == self.targetphase:216 **return** sol1217 **elif** sol2['total'] == self.targetphase:218 **return** sol2219 **elif** sol3['total'] == self.targetphase:220 **return** sol3221 **elif** sol4['total'] == self.targetphase:222 **return** sol4223 **else:**

```

    224         closest = min(sollist, key=lambda x: abs(
    225             dec.Decimal(x) - dec.Decimal(self.targetphase)))

```

226 **if** sol1['total'] == closest:227 **return** sol1228 **elif** sol2['total'] == closest:229 **return** sol2230 **elif** sol3['total'] == closest:231 **return** sol3232 **elif** sol4['total'] == closest:

```

233         return sol4
234     else:
235         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

```

1  """Module to test two phases."""
2  import decimal as dec
3  import os
4
5
6  def check1(self, set180, set90):
7      """Check 180 and 90 degrees."""
8      dec.getcontext().prec = 6
9      combined = set(map(str.rstrip, open(
10         (os.path.join(os.path.dirname(__file__), '18090.txt')))))
11      closest = min(combined, key=lambda x: abs(
12         dec.Decimal(x) - dec.Decimal(self.targetphase)))
13      closest = dec.Decimal(closest)
14      del combined
15      for i in set180:
16          for j in set90:
17              total = i + j
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 =
23                              ↪ dec.Decimal(row[self.att28].value).quantize(
24                                  dec.Decimal('.001'),
25                                  ↪ rounding=dec.ROUND_HALF_UP)
26                          phaselow1 =
27                              ↪ dec.Decimal(row[self.phase24].value).quantize(

```

```

25         dec.Decimal('.001'),
        ↪ rounding=dec.ROUND_HALF_UP)
26     phasehigh1 =
        ↪ dec.Decimal(row[self.phase32].value).quantize(
27         dec.Decimal('.001'),
        ↪ rounding=dec.ROUND_HALF_UP)
28     phasediff1 = phasehigh1 - phaselow1
29
30     for row in self.s90:
31         if row[self.phase28].value == j:
32             row2 = int(row[0].value)
33             att2 =
34                 ↪ dec.Decimal(row[self.att28].value).quantize(
35                 dec.Decimal('.001'),
36                 ↪ rounding=dec.ROUND_HALF_UP)
37             phaselow2 =
38                 ↪ dec.Decimal(row[self.phase24].value).quantize(
39                 dec.Decimal('.001'),
40                 ↪ rounding=dec.ROUND_HALF_UP)
41             phasehigh2 =
42                 ↪ dec.Decimal(row[self.att32].value).quantize(
43                 dec.Decimal('.001'),
44                 ↪ rounding=dec.ROUND_HALF_UP)
45             phasediff2 = phasehigh2 - phaselow2
46
47     totalatt = att1 + att2
48     return{'phase1': i.quantize(dec.Decimal('.001'),
49         ↪ rounding=dec.ROUND_HALF_UP), 'row1': row1, 'att1':
50         ↪ att1, 'phasediff1': phasediff1, 'source1': 's180',
51         ↪ 'phase2': j.quantize(dec.Decimal('.001'),
52         ↪ rounding=dec.ROUND_HALF_UP), 'row2': row2, 'att2':
53         ↪ att2, 'phasediff2': phasediff2, 'source2': 's90',
54         ↪ 'total': dec.Decimal(total), 'totalatt':
55         ↪ totalatt.quantize(dec.Decimal('.001'),
56         ↪ rounding=dec.ROUND_HALF_UP)}
57
58 def check2(self, set180, set45):
59     """Check 180 and 45 degrees."""
60     dec.getcontext().prec = 6
61     combined = set(map(str.rstrip, open(
62         (os.path.join(os.path.dirname(__file__), '18045.txt')))))
63     closest = min(combined, key=lambda x: abs(
64         dec.Decimal(x) - dec.Decimal(self.targetphase)))
65     closest = dec.Decimal(closest)

```

```

53     for i in set180:
54         for j in set45:
55             total = i + j
56             if total == closest:
57                 for row in self.s180:
58                     if row[self.phase28].value == i:
59                         row1 = int(row[0].value)
60                         att1 =
61                             ↳ dec.Decimal(row[self.att28].value).quantize(
62                                 dec.Decimal('.001'),
63                                 ↳ rounding=dec.ROUND_HALF_UP)
64                         phaselow1 =
65                             ↳ dec.Decimal(row[self.phase24].value).quantize(
66                                 dec.Decimal('.001'),
67                                 ↳ rounding=dec.ROUND_HALF_UP)
68                         phasehigh1 =
69                             ↳ dec.Decimal(row[self.phase32].value).quantize(
70                                 dec.Decimal('.001'),
71                                 ↳ rounding=dec.ROUND_HALF_UP)
72                         phasediff1 = phasehigh1 - phaselow1
73
74                 for row in self.s45:
75                     if row[self.phase28].value == j:
76                         row2 = int(row[0].value)
77                         att2 =
78                             ↳ dec.Decimal(row[self.att28].value).quantize(
79                                 dec.Decimal('.001'),
80                                 ↳ rounding=dec.ROUND_HALF_UP)
81                         phaselow2 =
82                             ↳ dec.Decimal(row[self.phase24].value).quantize(
83                                 dec.Decimal('.001'),
84                                 ↳ rounding=dec.ROUND_HALF_UP)
85                         phasehigh2 =
86                             ↳ dec.Decimal(row[self.phase32].value).quantize(
87                                 dec.Decimal('.001'),
88                                 ↳ rounding=dec.ROUND_HALF_UP)
89                         phasediff2 = phasehigh2 - phaselow2
90
91             totalatt = att1 + att2

```

```

80         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)}

81
82
83     def check3(self, set180, set225):
84         """Check 180 and 22.5 degrees."""
85         dec.getcontext().prec = 6
86         combined = set(map(str.rstrip, open(
87             (os.path.join(os.path.dirname(__file__), '180225.txt')))))
88         closest = min(combined, key=lambda x: abs(
89             dec.Decimal(x) - dec.Decimal(self.targetphase)))
90         closest = dec.Decimal(closest)
91         for i in set180:
92             for j in set225:
93                 total = i + j
94                 if total == closest:
95                     for row in self.s180:
96                         if row[self.phase28].value == i:
97                             row1 = int(row[0].value)
98                             att1 =
99                                 ↪ dec.Decimal(row[self.att28].value).quantize(
100                                     dec.Decimal('.001'),
101                                     ↪ rounding=dec.ROUND_HALF_UP)
102                             phaselow1 =
103                                 ↪ dec.Decimal(row[self.phase24].value).quantize(
104                                     dec.Decimal('.001'),
105                                     ↪ rounding=dec.ROUND_HALF_UP)
106                             phasehigh1 =
107                                 ↪ dec.Decimal(row[self.phase32].value).quantize(
108                                     dec.Decimal('.001'),
109                                     ↪ rounding=dec.ROUND_HALF_UP)
110                             phasediff1 = phasehigh1 - phaselow1
111
112                     for row in self.s225:
113                         if row[self.phase28].value == j:
114                             row2 = int(row[0].value)
115                             att2 =
116                                 ↪ dec.Decimal(row[self.att28].value).quantize(

```

```

110         dec.Decimal('.001'),
            ↳ rounding=dec.ROUND_HALF_UP)
111     phaselow2 =
            ↳ dec.Decimal(row[self.phase24].value).quantize(
112         dec.Decimal('.001'),
            ↳ rounding=dec.ROUND_HALF_UP)
113     phasehigh2 =
            ↳ dec.Decimal(row[self.phase32].value).quantize(
114         dec.Decimal('.001'),
            ↳ rounding=dec.ROUND_HALF_UP)
115     phasediff2 = phasehigh2 - phaselow2
116
117     totalatt = att1 + att2
118     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)}
119
120
121     def check4(self, set90, set45):
122         """Check 90 and 45 degrees."""
123         dec.getcontext().prec = 6
124         combined = set(map(str.rstrip, open(
125             (os.path.join(os.path.dirname(__file__), '9045.txt')))))
126         closest = min(combined, key=lambda x: abs(
127             dec.Decimal(x) - dec.Decimal(self.targetphase)))
128         closest = dec.Decimal(closest)
129         for i in set90:
130             for j in set45:
131                 total = i + j
132                 if total == closest:
133                     for row in self.s90:
134                         if row[self.phase28].value == i:
135                             row1 = int(row[0].value)
136                             att1 =
                                ↳ dec.Decimal(row[self.att28].value).quantize(
137                             dec.Decimal('.001'),
                                ↳ rounding=dec.ROUND_HALF_UP)
138                             phaselow1 =
                                ↳ dec.Decimal(row[self.phase24].value).quantize(

```

```

139         dec.Decimal('.001'),
140         ↪ rounding=dec.ROUND_HALF_UP)
141     phasehigh1 =
142         ↪ dec.Decimal(row[self.phase32].value).quantize(
143         dec.Decimal('.001'),
144         ↪ rounding=dec.ROUND_HALF_UP)
145     phasediff1 = phasehigh1 - phaselow1
146
147     for row in self.s45:
148         if row[self.phase28].value == j:
149             row2 = int(row[0].value)
150             att2 =
151                 ↪ dec.Decimal(row[self.att28].value).quantize(
152                 dec.Decimal('.001'),
153                 ↪ rounding=dec.ROUND_HALF_UP)
154             phaselow2 =
155                 ↪ dec.Decimal(row[self.phase24].value).quantize(
156                 dec.Decimal('.001'),
157                 ↪ rounding=dec.ROUND_HALF_UP)
158             phasehigh2 =
159                 ↪ dec.Decimal(row[self.phase32].value).quantize(
160                 dec.Decimal('.001'),
161                 ↪ rounding=dec.ROUND_HALF_UP)
162             phasediff2 = phasehigh2 - phaselow2
163
164     totalatt = att1 + att2
165     return{'phase1': i.quantize(dec.Decimal('.001'),
166         ↪ rounding=dec.ROUND_HALF_UP), 'row1': row1, 'att1':
167         ↪ att1, 'phasediff1': phasediff1, 'source1': 's90',
168         ↪ 'phase2': j.quantize(dec.Decimal('.001'),
169         ↪ rounding=dec.ROUND_HALF_UP), 'row2': row2, 'att2':
170         ↪ att2, 'phasediff2': phasediff2, 'source2': 's45',
171         ↪ 'total': total, 'totalatt':
172         ↪ totalatt.quantize(dec.Decimal('.001'),
173         ↪ rounding=dec.ROUND_HALF_UP)}
174
175
176 def check5(self, set90, set225):
177     """Check 90 and 22.5 degrees."""
178     dec.getcontext().prec = 6
179     combined = set(map(str.rstrip, open(
180         (os.path.join(os.path.dirname(__file__), '90225.txt')))))
181     closest = min(combined, key=lambda x: abs(
182         dec.Decimal(x) - dec.Decimal(self.targetphase)))
183     closest = dec.Decimal(closest)

```

```

167     for i in set90:
168         for j in set225:
169             total = i + j
170             if total == closest:
171                 for row in self.s90:
172                     if row[self.phase28].value == i:
173                         row1 = int(row[0].value)
174                         att1 =
175                             ↪ dec.Decimal(row[self.att28].value).quantize(
176                                 dec.Decimal('.001'),
177                                 ↪ rounding=dec.ROUND_HALF_UP)
178                         phaselow1 =
179                             ↪ dec.Decimal(row[self.phase24].value).quantize(
180                                 dec.Decimal('.001'),
181                                 ↪ rounding=dec.ROUND_HALF_UP)
182                         phasehigh1 =
183                             ↪ dec.Decimal(row[self.phase32].value).quantize(
184                                 dec.Decimal('.001'),
185                                 ↪ rounding=dec.ROUND_HALF_UP)
186                         phasediff1 = phasehigh1 - phaselow1
187
188                 for row in self.s225:
189                     if row[self.phase28].value == j:
190                         row2 = int(row[0].value)
191                         att2 =
192                             ↪ dec.Decimal(row[self.att28].value).quantize(
193                                 dec.Decimal('.001'),
194                                 ↪ rounding=dec.ROUND_HALF_UP)
195                         phaselow2 =
196                             ↪ dec.Decimal(row[self.phase24].value).quantize(
197                                 dec.Decimal('.001'),
198                                 ↪ rounding=dec.ROUND_HALF_UP)
199                         phasehigh2 =
200                             ↪ dec.Decimal(row[self.phase32].value).quantize(
201                                 dec.Decimal('.001'),
202                                 ↪ rounding=dec.ROUND_HALF_UP)
203                         phasediff2 = phasehigh2 - phaselow2
204
205             totalatt = att1 + att2

```



```

194         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)}

195
196
197     def check6(self, set45, set225):
198         """Check 45 and 22.5 degrees."""
199         dec.getcontext().prec = 6
200         combined = set(map(str.rstrip, open(
201             (os.path.join(os.path.dirname(__file__), '45225.txt')))))
202         closest = min(combined, key=lambda x: abs(
203             dec.Decimal(x) - dec.Decimal(self.targetphase)))
204         closest = dec.Decimal(closest)
205         for i in set45:
206             for j in set225:
207                 total = dec.Decimal(i) + dec.Decimal(j)
208                 if total == closest:
209                     for row in self.s45:
210                         if row[self.phase28].value == i:
211                             row1 = int(row[0].value)
212                             att1 =
213                                 ↪ dec.Decimal(row[self.att28].value).quantize(
214                                     dec.Decimal('.001'),
215                                     ↪ rounding=dec.ROUND_HALF_UP)
216                             phaselow1 =
217                                 ↪ dec.Decimal(row[self.phase24].value).quantize(
218                                     dec.Decimal('.001'),
219                                     ↪ rounding=dec.ROUND_HALF_UP)
220                             phasehigh1 =
221                                 ↪ dec.Decimal(row[self.phase32].value).quantize(
222                                     dec.Decimal('.001'),
223                                     ↪ rounding=dec.ROUND_HALF_UP)
224                             phasediff1 = phasehigh1 - phaselow1
225
226             for row in self.s225:
227                 if row[self.phase28].value == j:
228                     row2 = int(row[0].value)
229                     att2 =
230                         ↪ dec.Decimal(row[self.att28].value).quantize(

```

```

224         dec.Decimal('.001'),
           ↳ rounding=dec.ROUND_HALF_UP)
225     phaselow2 =
           ↳ dec.Decimal(row[self.phase24].value).quantize(
226         dec.Decimal('.001'),
           ↳ rounding=dec.ROUND_HALF_UP)
227     phasehigh2 =
           ↳ dec.Decimal(row[self.phase32].value).quantize(
228         dec.Decimal('.001'),
           ↳ rounding=dec.ROUND_HALF_UP)
229     phasediff2 = phasehigh2 - phaselow2
230
231     totalatt = att1 + att2
232     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)}
233
234
235     def checkall(self, set180, set90, set45, set225):
236         """Check for most accurate two phase solution."""
237         dec.getcontext().prec = 6
238         sollist = []
239         sol1 = check1(self, set180, set90)
240         sollist.append(sol1['total'])
241         sol2 = check2(self, set180, set45)
242         sollist.append(sol2['total'])
243         sol3 = check3(self, set180, set225)
244         sollist.append(sol3['total'])
245         sol4 = check4(self, set90, set45)
246         sollist.append(sol4['total'])
247         sol5 = check5(self, set90, set225)
248         sollist.append(sol5['total'])
249         sol6 = check6(self, set45, set225)
250         sollist.append(sol6['total'])
251
252         if sol1['total'] == self.targetphase:
253             return sol1
254         elif sol2['total'] == self.targetphase:
255             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):

```

```

14     for j in s90.iter_rows(row_offset=2):
15         print(j)
16         if i[2].value is not None and j[2].value is not None:
17             convert1 = dec.Decimal(i[2].value)
18             convert2 = dec.Decimal(j[2].value)
19             total = convert1 + convert2
20             total = total.quantize(dec.Decimal('.001'),
21                                     ↪ rounding=dec.ROUND_HALF_UP)
21             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

```

1  """Format numbers to be output as settings for MPAC."""
2
3
4  def phaseformat(self, bestphase):
5      """Format phase settings correctly."""
6      s180present = 0
7      s90present = 0
8      s45present = 0
9      s225present = 0
10     if 'source1' in bestphase:
11         if bestphase['source1'] == 's180':
12             s180present = 1
13         elif bestphase['source1'] == 's90':
14             s90present = 1
15         elif bestphase['source1'] == 's45':
16             s45present = 1
17         elif bestphase['source1'] == 's225':
18             s225present = 1
19     if 'source2' in bestphase:
20         if bestphase['source2'] == 's90':
21             s90present = 2
22         elif bestphase['source2'] == 's45':
23             s45present = 2
24         elif bestphase['source2'] == 's225':
25             s225present = 2
26
27     if 'source3' in bestphase:
28         if bestphase['source3'] == 's45':
29             s45present = 3
30         elif bestphase['source3'] == 's225':

```

```

31         s225present = 3
32
33     if 'source4' in bestphase:
34         if bestphase['source4'] == 's225':
35             s225present = 4
36     if s180present == 1:
37         s180setting = '{0:02b}'.format(bestphase['row1'])
38     else:
39         s180setting = '{0:02b}'.format(2)
40
41     if s90present == 1:
42         s90setting = '{0:06b}'.format(bestphase['row1'])
43     elif s90present == 2:
44         s90setting = '{0:06b}'.format(bestphase['row2'])
45         s90present = 1
46     else:
47         s90setting = '{0:06b}'.format(32)
48
49     if s45present == 1:
50         s45setting = '{0:09b}'.format(bestphase['row1'])
51     elif s45present == 2:
52         s45setting = '{0:09b}'.format(bestphase['row2'])
53         s45present = 1
54     elif s45present == 3:
55         s45setting = '{0:09b}'.format(bestphase['row3'])
56         s45present = 1
57     else:
58         s45setting = '{0:09b}'.format(256)
59
60     if s225present == 1:
61         s225setting = '{0:09b}'.format(bestphase['row1'])
62     elif s225present == 2:
63         s225setting = '{0:09b}'.format(bestphase['row2'])
64         s225present = 1
65     elif s225present == 3:
66         s225setting = '{0:09b}'.format(bestphase['row3'])
67         s225present = 1
68     elif s225present == 4:
69         s225setting = '{0:09b}'.format(bestphase['row4'])
70         s225present = 1
71     else:
72         s225setting = '{0:09b}'.format(256)
73
74     return {'s180setting': s180setting, 's90setting': s90setting,
75           ↪ '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.