Appendix

Table of Contents

1.1 Appendix – Chapter 3

1.1.1 Chapter 3 – Appendix 1: Chimera Script to prepare ligand files for Hungarian algorithm

```
#!/usr/bin/python

'''
retrieve docking solutions and use chimera to
prepare files for input into RMSD script
'''

from chimera import runCommand as rc
from chimera import replyobj
import os
import re
import sys

flexible = True  #is the docking run flexible
consensus_level = 2.0 #the level of consensus you want to search
for in Angstroms

#location of vina output results
vina_path =
"/Users/nj001/Documents/Docking/201702/20170221/VINA/20170221_2_A
bb/"

#location of gold output results
gold_path =
"/Users/nj001/Documents/Docking/201702/20170221/GOLD/20170221_2_A
bb/"

#where to put results
out_dir =
"/Users/nj001/Documents/Docking/201702/20170221/clusters/20170221
_2_Abb/"
#os.mkdir(out_dir)

vina_out_directories = []
dirs = []
open_models = 0

make_GvA = 'T' #raw_input('Write GvA files? (T/F) --->')

#-----------------------------------------------------------------
-------------

def alphanum_keys(s):
      num = s[:-5].split("_")
      item = num.pop(-1)
      return int(item)

def sort_nicely(l): #sort by natural sort
      keys = []
      for item in l:
            key = alphanum_keys(item)
            keys.append(key)
```

```
      file_keys = zip(l, keys)
      return sorted(file_keys, key = lambda x:x[1])

#-----------------------------------------------------------------
-------------

vina_out_directories = [fn for fn in next(os.walk(vina_path))[1]
if fn.startswith('ligand')]
gold_out_directories = [fn for fn in next(os.walk(gold_path))[1]
if fn.endswith('_m1')]

gold_out_directories_2 = [re.sub('.pdb','',x) if '.pdb' in x else
x for x in gold_out_directories]

#vina_out_directories = sort_nicely(vina_out_directories)
#gold_out_directories = sort_nicely(gold_out_directories)

print vina_out_directories
print gold_out_directories

for item in vina_out_directories:
      if (item[7:]+'_m1') in gold_out_directories_2:
            continue
      else:
            print item,'not in GOLD solutions'
            vina_out_directories.remove(item)

print vina_out_directories
print gold_out_directories

dirs.append(vina_out_directories)
dirs.append(gold_out_directories)

print dirs

#go through each vina out.pdbqt and write a mol2 file
if make_GvA == 'T':
      for directory in vina_out_directories:
            if flexible == True:
                  replyobj.status("Processing " + directory) #what
are we working on
                  f = (vina_path + directory + "/out.pdbqt")
                  rc("viewdock " + f) #open pdbqt files
                  rc("sel
#0:phe,ile,leu,asn,met,thr,tyr,ser,glu,lys") #select flexible
sidechains
                  rc("delete sel") #remove the atoms
                  rc("addh spec #0") #add all hydrogens
            elif flexible == False:
                  replyobj.status("Processing " + directory) #what
are we working on
                  f = (vina_path + directory + "/out.pdbqt")
                  rc("viewdock " + f) #open pdbqt files
                  rc("addh spec #0") #add all hydrogens
            for i in range(1,21): #iterate through open models
(max 20 for vina)
                  rc("sel #0."+str(i))
```

```
                #write .mol2 for each model will write empty
atom files if not 20 models
                rc("write format mol2 selected relative 0.1 #0
"+vina_path+directory+"/"+directory+"_"+str(i)+".mol2")
          rc("close session") #close everything so that in next
iteration the model number stays at 0

     for i in range(len(vina_out_directories)):
          os.chdir(vina_path+"/"+dirs[0][i]) #go to vina
directory
          files = [f for f in os.listdir(".") if
f.endswith(".mol2")]
          print files
          files = sort_nicely(files)
          print files
          for f in files:
               fo = open(f[0],"r")
               for x, line in enumerate(fo): #remove files that
contain no atoms
                    if x == 2: #third line of .mol2 reads '0 0
0 0' if no atoms in file
                         ln = line.split()
                         if ln[0] == "0":
                              os.remove(f[0])
                         else:
                              rc("open "+f[0]) #open all vina
results written to .mol2 files
                              open_models += 1
               fo.close()
          os.chdir(gold_path+dirs[1][i])
          files = [f for f in os.listdir(".") if
f.startswith("gold_soln_")] #go to gold directory
          print files
          files = sort_nicely(files)
          print files
          for f in files:
               rc("open "+f[0]) #open all gold solutions
labelled as they were run
               open_models += 1
          rc("select")
          rc("addh spec sel") #remove lone pairs on gold
solutions
          if os.path.isdir(out_dir+dirs[1][i]) == False: #check
to see is directory for output already exists
               os.mkdir(out_dir+dirs[1][i])
          for x in range(open_models):
               rc("sel #"+str(x))
               rc("write format mol2 selected relative 0
"+str(x)+" "+out_dir+dirs[1][i]+"/"+dirs[1][i][:-
2]+str(x+1)+".mol2") #write .mol2 for each model
          rc("close session")
          os.chdir(out_dir+dirs[1][i])
          files = [f for f in os.listdir(".") if
f.endswith(".mol2")]
          files = sort_nicely(files)
          for f in files: #remove any defunct files again after
writing all the gold and vina files to one directory
               fo = open(f[0],"r")
```

```
                for x, line in enumerate(fo):
                        if x==2:
                                ln =line.split()
                                if ln[0] == "0":
                                        os.remove(f[0])
                fo.close()
        #concatenate all the files into one GvA.mol2 file
        fname = (dirs[1][i][:-2]+"GvA.mol2")
        outfile = open((out_dir+fname),"w")
        for f in files:
                with open(f[0],'r') as infile:
                        outfile.write(infile.read())
        outfile.close()
        open_models = 0

os.system('/usr/bin/python
~/Documents/Docking/ANALYSIS/docking_soln_analysis_2.py
'+vina_path+' '+gold_path+' '+out_dir)
```

1.1.2 Chapter 3 – Appendix 2: Python Script to determine RMSD using Hungarian algorithm

```
#usr/bin/python

import sys
import os
import scipy
from scipy.cluster import hierarchy
import numpy as np
from munkres import Munkres, print_matrix
from math import sqrt
import matplotlib.pyplot as plt
import itertools
import operator
#from Tkinter import Tk
#from tkFileDialog import askdirectory
import pylab
import re

#atom types in SYBYL MOL2 format
atom_types = ['C.3','C.2','C.ar','N.pl3','O.2','S.o2',
      'C.1','N.1','S.3','S.O','P.3','N.4','O.co2','S.2',
      'P.3','N.3','O.3']


consensus_level = 2.5 #the level of consensus you want to search
for in Angstroms

#location of vina output results
vina_path = sys.argv[1]
#"/Users/nj001/Documents/Chemistry/Open_Eye/ALLOSTERIC_AGONISTS/P
DBQT_FILES/"

#location of gold output results
gold_path = sys.argv[2]
#"/Users/nj001/Documents/Chemistry/Open_Eye/ALLOSTERIC_AGONISTS/G
OLD_RESULTS/"

#where to put results
out_dir = sys.argv[3]
#"/Users/nj001/Documents/Chemistry/Open_Eye/ALLOSTERIC_AGONISTS/c
luster_analysis/"
#os.mkdir(out_dir)

flexible = bool(sys.argv[4])  #is the docking run flexible

number_of_solns = 5 #number of consensus solutions to be included
in hierarchical clustering
rmsd_cut_off = 2.5 #distance cutoff for clustering of ligand
solutions against each other

vina_out_directories = []
dirs = []
open_models = 0

make_rmsd_matrix = 'T' #raw_input('Write RMSD matrix? (T/F) ---
>')
```

```
consensus_scoring = 'T' #raw_input('Calculate and write Borda
ranks? (T/F) --->')
create_interligand_mol2 = 'T' #raw_input('Make mol2 with
consensus solutions? (T/F) --->')
make_all_by_all_matrix = 'T' #raw_input('Write all by all RMSD
matrix? (T/F) --->')
write_clusters = 'T' #raw_input('Write the clusters into files?
(T/F) --->')
run_chimera = 'T' #raw_input('Make chimera session of largest
cluster? (T/F) --->')

#----------------------------------------------------------------
------

def make_matrices(file_name, first_time):
      '''
      take a GvA file and make list of lists of atom coordinates
      returned structure:
      [[[atom[atom_idx,x,y,z,atom_type]], mol_name, atom_count]]
      '''
      parsing = False
      in_molecule = False
      dictionary = {}
      atom_list = []
      mol_list = []
      atom_count = 0
      molecule_count = 0
      vina_count = 0
      name_counter = 0
      with open(file_name) as f:
            lines = f.readlines()
            for line in lines:
                  #read only the relevant lines in the file
(parsing between the
                  #start of atom list and start of bond list)
                  if line.startswith("@<TRIPOS>BOND"):
                        parsing = False
                        #make a dictionary of the dictionaries of
coordinates
                        mol_list.append([atom_list,mol_name,
atom_count])
                        atom_list = []
                        molecule_count += 1
                        #total_atoms = atom_count
                        atom_count = 0
                  if parsing:
                        ln = line.split()
                        coords = []
                        #add the coordinates to a dictionary with
the atom number as the key
                        atom_idx = int(ln[0])
                        atom_type = ln[5]
                        x = float(ln[2])
                        y = float(ln[3])
                        z = float(ln[4])
                        if atom_type != 'H':
                              coords = [atom_idx, x, y, z,
atom_type]
```

```
                              atom_list.append(coords)
                              atom_count += 1
                  if line.startswith("@<TRIPOS>ATOM"):
                        parsing = True
                  if line.startswith("@<TRIPOS>MOLECULE"):
                        in_molecule = True
                        continue
                  if in_molecule == True:
                        if first_time == True:
                              mol_name = file_name[:-
9]+'_'+str(molecule_count+1) #(mol[1][:-4]+'_'+mol[4][4:-1])
                        if first_time == False:
                              mol = line.split('|')
                              #print mol
                              mol_name = (mol[1]+'_m1_'+mol[4][4:])
                        in_molecule = False
            #vina_count = molecule_count - 50
            #ligand = file_name[:-10]
            #mols_list = [mol_list, molecule_count] #ligand,
vina_count, total_atoms,
      return mol_list  #[]

#-----------------------------------------------------------------
------

def hungarian_alg(matrix):
      '''
      runs hungarian algorithm to determine "lowest cost"
assignment
      of atoms, for the calculation of RMSD
      very slow to only use if you have to
      '''
      m=Munkres()
      indexes = m.compute(matrix)
      total = 0
      matched_atoms = 0
      for row, column in indexes:
            value = matrix[row][column]
            total += value
            matched_atoms += 1
            #print '(%d, %d) -> %d' % (row, column, value)
      return [total, matched_atoms]

#-----------------------------------------------------------------
------

def rmsd_matrix(list1, list2):
      '''
      creates all atom against all atom RMSD matrix
      for input to hungarian algorithm
      list format is [atom[atom_idx,x,y,z,atom_type]]
      '''
      j_list = []
      rmsds = []
      index = 0
      for i in range(len(list1)):
            for j in range(len(list2)):
                  for c in (1,2,3):
```

```
                        index += (list2[j][c]-list1[i][c]) ** 2
                  j_list.append(index)
                  index = 0
            rmsds.append(j_list)
            j_list = []
      return rmsds

#-----------------------------------------------------------------
------

def f7(seq):
      '''
      removes duplicates from list
      used here to ensure molecules which come in top 5 ranks
      more than once are only accounted for once
      '''
      seen = set()
      seen_add = seen.add
      return [x for x in seq if not(x in seen or seen_add(x))]

#-----------------------------------------------------------------
------

def condense(atom_type, list):
      '''
      shortens list of atoms into atom types to form smaller
matrices
      for hungarian algorithm.  The specificity of only matching
the
      atoms of the same type also affords correct asisgnment using
      the hungarian algorithm
      '''
      condensed_list = []
      for item in list:
            if item[4] == atom_type:
                  #print item[4], atom_type
                  condensed_list.append(item)
      return condensed_list

#-----------------------------------------------------------------
------

def sort_nicely(l): #sort by natural sort
      keys = []
      for item in l:
            key = alphanum_keys(item)
            keys.append(key)
      file_keys = zip(l, keys)
      return sorted(file_keys, key = lambda x:x[1])

#-----------------------------------------------------------------
------

def ranking(ordered_list):
      ranks = [1]
      for i in range(1, len(ordered_list)):
            if ordered_list[i][0] == ordered_list[i-1][0]:
                  ranks.append(ranks[i-1])
```

```
            else:
                  ranks.append(len(ranks)+1)
      return ranks

#-----------------------------------------------------------------
------

def get_vina_rank(consensus_list): #WORKING
      numbering = range(1,len(consensus_list)+1)
      vina_list = zip(consensus_list, numbering)  #assign order by
rmsd to solution numbers
      vina_list = sorted(vina_list, key=lambda tuple: tuple[0])
#sort by vina score
      ranks = ranking(vina_list)#[1]
      vina_ranks = zip(vina_list,ranks)
      vina_ranks = sorted(vina_ranks, key = lambda tuple:
tuple[0][1])  #reorder by rmsd rank so that the correct solutions
can be related back to the gold solns
      return vina_ranks

#-----------------------------------------------------------------
------

def get_goldscore(consensus_list,file): #WORKING
      gold_scores_list = []
      numbering = range(1,len(consensus_list)+1)
      gold_list = zip(consensus_list, numbering)  #assign order by
rmsd to solution numbers
      with open(file) as f:
            for i, line in enumerate(f):
                  if i > 3 and i < 54:     #only read the relevant
section of the file
                        ln = line.split()
                        for item in gold_list:
                              print item
                              print item[0]
                              print ln[0]
                              if item[0] == int(ln[0]):

      gold_scores_list.append([item[0], float(ln[1]), item[1]])
#[gold_soln_num, gold_score, rmsd_rank]
            gold_score_solns_sorted = sorted(gold_scores_list, key
= lambda tuple: tuple[1], reverse=True)
            ranks = ranking(gold_score_solns_sorted)
            gold_score_ranks = zip(gold_score_solns_sorted, ranks)
            gold_score_ranks = sorted(gold_score_ranks, key =
lambda tuple: tuple[0][2])
            return gold_score_ranks

#-----------------------------------------------------------------
------

def get_chemplp(consensus_list,sorted_rmsd_name,rescore_file,
flexible): #WORKING
      print "<<getting chemplp>>",sorted_rmsd_name[15:-4]
      chem_plp_list = []
      numbering = range(1,len(consensus_list)+1)
      gold_chemplp_list = zip(consensus_list, numbering)
```

```
      print "<<CONSENSUS_LIST>>\n",gold_chemplp_list
      with open(rescore_file) as f:
            for i, line in enumerate(f):
                  if i > 2:
                        ln = line.split()
                        if flexible == True:
                              identifier = ln[12]
                        else:
                              identifier = ln[11]
                        fitness_score = float(ln[3])
                        broken_ids = identifier.split('|')
                        mol_name = broken_ids[1]
                        dock_num = int(broken_ids[4][4:])
                        print "<<Docking run number>>",dock_num
                        if sorted_rmsd_name[15:-4] == mol_name:
                              for item in gold_chemplp_list:
                                    if item[0] == dock_num:
                                          print fitness_score

      chem_plp_list.append([item[0], fitness_score, item[1]])
      #[gold_number, chem_plp_score, rmsd_rank]
            chem_plp_sorted = sorted(chem_plp_list, key = lambda
tuple:tuple[1], reverse=True)
            ranks = ranking(chem_plp_sorted)
            chem_plp_ranks = zip(chem_plp_sorted, ranks)
            chem_plp_ranks = sorted(chem_plp_ranks, key = lambda
tuple: tuple[0][2])
            return chem_plp_ranks

#-------------------------------------------------------------------
------

def borda_score(total_number, ranks):
      bordascore = 0
      for i in range(len(ranks)):
            bordascore += total_number - ranks[i]
      return bordascore

#-------------------------------------------------------------------
------

def get_soln_numbers(file): #WORKING
    vina = []
    gold = []
    with open(file) as f:
            lines = f.readlines()
            for line in lines:
                  if line.startswith('#'):
                        continue
                  else:
                        ln = line.split()
                        RMSD = float(ln[2])
                        vina_soln_no = int(ln[0])
                        gold_soln_no = int(ln[1])
                  if RMSD <= 2.0:
                        vina.append(vina_soln_no)
                        gold.append(gold_soln_no)
            vina_gold = zip(vina,gold)
```

```
            if len(vina_gold) == 0:
                  return 'ERROR: NO SOLNS < 2.0 A RMSD'
            else:
                  return vina_gold

#-----------------------------------------------------------------
------

def check_vina_gold(vina_gold):
      for tuple in vina_gold:
            print tuple
            dup_list = [tup for tup in vina_gold if tup[1] ==
tuple[1]]
            if len(dup_list) > 1:
                  dup_list_sorted = sorted(dup_list, key = lambda
tuple: tuple[0])
                  for i in range(len(dup_list_sorted)):
                        if i > 0:
                              vina_gold.remove(dup_list[i])
      return vina_gold

#-----------------------------------------------------------------
------

def sort_by_cluster(list, cluster_num):
      '''
      sorts output of fcluster into a list of molecules in each
cluster
      '''
      cluster = []
      for item in list:
            if item[1] == cluster_num:
                  cluster.append(item)
      return cluster

#-----------------------------------------------------------------
------

def file_len(fname):
    with open(fname) as f:
        for i, l in enumerate(f):
            pass
    return i + 1

#-----------------------------------------------------------------
------
#################################################################
######
os.chdir(out_dir)
GvAs = [f for f in os.listdir(".") if f.endswith("GvA.mol2")]

#### Calculate RMSDS ####

for GvA in GvAs:
      molecules = make_matrices(out_dir+GvA, True)

      if make_rmsd_matrix == 'T':
            molecule_rmsds = []
```

```
            condensed_rmsds = []
            rmsd_square_matrix = []
            rmsds = []
            print len(molecules)
            for i,item in enumerate(molecules[:(len(molecules)-
50)]):
                  #print item
                  for j,item2 in
enumerate(molecules[(len(molecules)-50):]):
                        lowest_cost = 0
                        matched_atoms = 0
                        for atom_type in atom_types:
                              condensed_1 =
condense(atom_type,item[0])
                              condensed_2 =
condense(atom_type,item2[0])
                              matey =
rmsd_matrix(condensed_1,condensed_2)
                              if len(matey) > 0:
                                    #print matey
                                    hungry = hungarian_alg(matey)
                                    #print hungry
                                    lowest_cost += hungry[0]
                                    matched_atoms += hungry[1]
                        rmsd = sqrt(lowest_cost/matched_atoms)
                        molecule_rmsds.append(rmsd)
                        if rmsd < 2.0:
                              rmsds.append([i+1,j+1,rmsd])
                  #print molecule_rmsds
                  #condensed_rmsds = non_redundant(molecule_rmsds)

                  rmsd_square_matrix.append(molecule_rmsds)
                  molecule_rmsds = []
            with open(out_dir+'rmsd_'+GvA+'_matrix.txt','w') as
outfile:
                  for item in rmsd_square_matrix:
                        for rmsd in item:
                              outfile.write(str(rmsd)+'\t')
                        outfile.write('\n')
            sorted_rmsds = sorted(rmsds,key = lambda tuple:
tuple[2])
            print GvA[:-9]
            print 'VINA\tGOLD_soln\tRMSD'
            for item in sorted_rmsds:
                  print
(str(item[0])+'\t'+str(item[1])+'\t'+str(item[2]))
            with open((out_dir+'sorted_munkres_'+GvA[:-
9]+'.txt'),'w') as outfile:
                  outfile.write('#'+GvA[:-
9]+'\n#VINA\tGOLD_soln\tRMSD\n')
                  for item in sorted_rmsds:

      outfile.write(str(item[0])+'\t'+str(item[1])+'\t'+str(item[2
])+'\n')
      else:
            with
open(out_dir+'rmsd_'+str(number_of_solns)+'matrix.txt','r') as
infile:
```

```
                  rmsd_square_matrix=[]
                  for line in infile:
                        ln = line[:-1].split()
                        #print ln
                        line_lis = []
                        for item in ln:
                              #print item
                              line_lis.append(float(item))
                        rmsd_square_matrix.append(line_lis)

      rmsd_square_matrix = np.array(rmsd_square_matrix)

#### Get scores and perform borda analysis ####

if consensus_scoring == 'T':
      chem_plp_log = "rescore.log"

      sorted_rmsd_files = [f for f in os.listdir(out_dir) if
f.startswith('sorted_')]
      gold_ligand_dirs = next(os.walk(gold_path))[1]

      os.chdir(out_dir)

      for file in sorted_rmsd_files:
            print ('#'+file[15:-4])
            vina_gold = get_soln_numbers(file) #[vina_soln_num,
gold_soln_num]
            print vina_gold
            if type(vina_gold) is list:
                  vina_gold = check_vina_gold(vina_gold)
                  print vina_gold
                  vina_consensus = [item[0] for item in vina_gold]
                  vina_ranks = get_vina_rank(vina_consensus)
                  print 'VINA ===> ',vina_ranks # ((vina_number,
order_by_rmsd), rank)
                  gold_consensus = [item[1] for item in vina_gold]
                  print 'GOLD Consensus solutions
==>',gold_consensus
                  gold_score_ranks = get_goldscore(gold_consensus,
(gold_path+file[15:-4]+'_m1/'+file[15:-4]+'_m1.rnk'))
                  print 'GOLD_SCORE ===> ',gold_score_ranks
#((gold_number, goldscore, order_by_rmsd) rank)
                  chem_plp_ranks = get_chemplp(gold_consensus,
file, (gold_path+'rescore.log'),flexible)
                  print 'CHEM_PLP ===> ',chem_plp_ranks

                  VINA_GS_PLP = []  #combine the ranks of all the
scoring functions into a matrix (list of lists) columns = scoring
function, rows = solution
                  borda_scores = []
                  total_consensus_solutions = len(vina_ranks)

                  print
len(vina_ranks),len(gold_score_ranks),len(chem_plp_ranks)
                  VINA_GS_PLP = zip([x[1] for x in
vina_ranks],[y[1] for y in gold_score_ranks],[z[1] for z in
chem_plp_ranks])
```

```
                print VINA_GS_PLP

                for j in range(len(VINA_GS_PLP)):
                      score =
borda_score(total_consensus_solutions, VINA_GS_PLP[j])
                      borda_scores.append(score)
                      print score
                VINA_GOLD_BORDA =
zip(vina_consensus,gold_consensus,borda_scores)
                ranked_solutions = sorted(VINA_GOLD_BORDA, key =
lambda tuple: tuple[2], reverse=True)
                print "CHECK_VGB ==>",VINA_GOLD_BORDA
                with open((out_dir+'borda_ranked_'+file[15:-
4]+'.txt'),'w') as outfile:
                      outfile.write('#'+file[15:-4]+'\n')
                      outfile.write('#vina gold borda \n')
                      for i in range(len(ranked_solutions)):
                            for j in range(3):

      outfile.write(str(ranked_solutions[i][j])+' ')
                            outfile.write('\n')
          else:
                print 'NO CONSENSUS SOLUTIONS FOR FILE ', file

#### Cluster the consensus solutions between different ligands
####

os.chdir(out_dir)

rank_files = [f for f in os.listdir('.') if
f.startswith('borda_ranked_')]

all_top_solns = []

if create_interligand_mol2 == 'T':
      for file in rank_files:
          top_gold = []
          with open(file) as infile:
                for i, line in enumerate(infile):
                      if i > 1 and i < 2+number_of_solns:
                            ln = line.split()
                            top_gold.append(file[13:-
4]+'_m1_'+str(ln[1])+'.mol2')
                      elif i > 1+number_of_solns:
                            break
          gold_soln_dir_name = (file[13:-4]+'_m1/')
          top_gold = f7(top_gold)
          for soln in top_gold:
                all_top_solns.append(soln)
                #gold_file_name = ('gold_soln_'+file[13:-
4]+'.pdb_m1_'+soln+'.mol2')

      with
open(out_dir+'top_'+str(number_of_solns)+'_solns_all.mol2','w')
as outfile:
          for item in all_top_solns:
                print item
                parts = item.split('_')
```

```
                print parts
                m1_idx = parts.index('m1')
                name = '_'.join(parts[0:(m1_idx+1)])
                with open(gold_path+name+'/gold_soln_'+item) as
infile:
                      for line in infile:
                            if 'P-T' in line:
                                  line=re.sub('P-T','P_T',line)
#match all occurences
                                  outfile.write(line)
                            elif 'U-1' in line:
                                  line=re.sub('U-1','U_1',line)
#match all occurences
                                  outfile.write(line)
                            else:
                                  outfile.write(line)

#### RMSD calculation for clustering ####

molecules =
make_matrices(out_dir+'top_'+str(number_of_solns)+'_solns_all.mol
2', False)

if make_all_by_all_matrix == 'T':
      molecule_rmsds = []
      condensed_rmsds = []
      rmsd_square_matrix = []

      for item in molecules:
            #print item
            for item2 in molecules:
                  lowest_cost = 0
                  matched_atoms = 0
                  for atom_type in atom_types:
                        condensed_1 = condense(atom_type,item[0])
                        condensed_2 = condense(atom_type,item2[0])
                        matey =
rmsd_matrix(condensed_1,condensed_2)
                        if len(matey) > 0:
                              #print matey
                              hungry = hungarian_alg(matey)
                              #print hungry
                              lowest_cost += hungry[0]
                              matched_atoms += hungry[1]
                  rmsd = sqrt(lowest_cost/matched_atoms)
                  molecule_rmsds.append(rmsd)
            #print molecule_rmsds
            #condensed_rmsds = non_redundant(molecule_rmsds)

            rmsd_square_matrix.append(molecule_rmsds)
            molecule_rmsds = []
      with
open(out_dir+'rmsd_'+str(number_of_solns)+'matrix.txt','w') as
outfile:
            for item in rmsd_square_matrix:
                  for rmsd in item:
                        outfile.write(str(rmsd)+'\t')
                  outfile.write('\n')
```

```
else:
      with
open(out_dir+'rmsd_'+str(number_of_solns)+'matrix.txt','r') as
infile:
            rmsd_square_matrix=[]
            for line in infile:
                  ln = line[:-1].split()
                  #print ln
                  line_lis = []
                  for item in ln:
                        gold_path        #print item
                        line_lis.append(float(item))
                  rmsd_square_matrix.append(line_lis)

rmsd_square_matrix =np.array(rmsd_square_matrix)

print rmsd_square_matrix
z= hierarchy.linkage(rmsd_square_matrix, 'single')
#clusts =
hierarchy.fcluster(z,number_of_clusters,criterion='maxclust')
clusts = hierarchy.fcluster(z,rmsd_cut_off, criterion='distance')
print clusts

file_names = []
for item in molecules:
      print item[1]
      file_names.append(item[1])

clustered = zip(file_names,clusts)

for i in range(1,max(clusts)+1):
      cluster = sort_by_cluster(clustered,i)
      #representative = representative_class(cluster)
      #print 'Cluster',i,'representative class
=',representative,'of',len(cluster),'solutions\n'
      if len(cluster) > 2:
            print cluster #representative[0], cluster
            #for item in representative[1]:
            #     print item
            if write_clusters == 'T':
                  with
open(out_dir+'cluster_'+str(cluster[0][1])+'.txt','w') as
file_names_file:
                        for item in cluster:
                              pieces = item[0].split('_')
                              gold_file_name =
'_'.join(['gold_soln']+pieces[:-1]+[pieces[-1][:-1]+'.mol2'])
                              gold_dir_name = '_'.join(pieces[:-1])
                              print gold_dir_name, gold_file_name

      file_names_file.write(gold_dir_name+'/'+gold_file_name+'\n')

cluster_files = [f for f in os.listdir('.') if
f.startswith('cluster') and f.endswith('txt')]
largest_cluster = ''
cluster_size = 0
for file in cluster_files:
      clus_length = file_len(file)
```

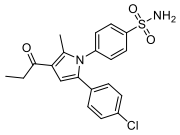
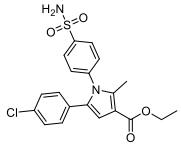
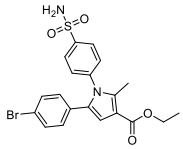
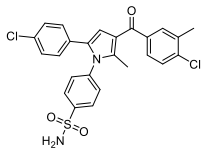
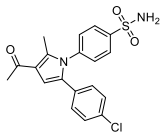
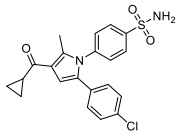
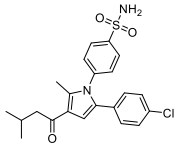
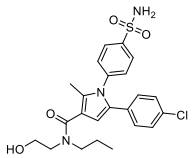
```
      print 'Number in cluster '+file+' ===> ',clus_length
      if clus_length > cluster_size:
            cluster_size = clus_length
            largest_cluster = file

print 'Largest cluster is ====>',largest_cluster
'''
if run_chimera == 'T':
      os.system('/Applications/Chimera.app/Contents/Resources/bin/
chimera --nogui --script
"~/Documents/Docking/ANALYSIS/open_clusters.py '+out_dir+'
'+gold_path+' '+largest_cluster+'"')
'''
for file in cluster_files:
      os.system('/Applications/Chimera.app/Contents/Resources/bin/
chimera --nogui --script
"~/Documents/Docking/ANALYSIS/open_clusters.py '+out_dir+'
'+gold_path+' '+file+'"')
```

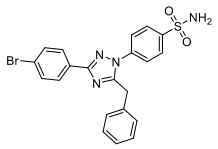
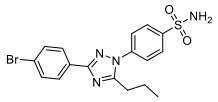
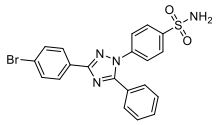
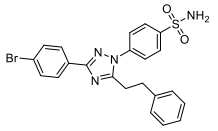
1.1.3 Chapter 3 – Appendix 3: RDKit 3D file generator from smiles

```
from rdkit import Chem
from rdkit.Chem import AllChem
import sys

#get smiles suppl
smiles_f = sys.argv[1]
smiles = []
codes = []
with open(smiles_f,'r') as smi_f:
      for line in smi_f:
            ln = line.split()
            smiles.append(ln[0])
            codes.append(ln[1])

#iterate through smiles creating 3D mols and appending to list
embedded_mols = [] # 3D molecule representations to get written
to file
cann_smiles = [] #list of canonical smiles that are put forward
for docking
names = [] #unique names for molecules
for i in range(0,len(smiles)):
      # check smiles are read correctly before making mol instance
      if Chem.MolFromSmiles(smiles[i]) == None:
            print "Molecule Not Read: %s" % smiles[i]
            continue
      m = Chem.MolFromSmiles(smiles[i])

      # check molecule is unique and not already read
      if smiles[i] in cann_smiles:
            print "Smiles already read: %s" % smiles[i]
            continue

      # check that the name is unique
      if codes[i] in names:
            print "Code already used: %s" % codes[i]
```

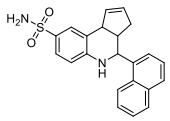
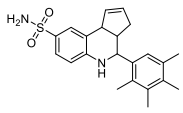
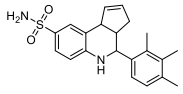
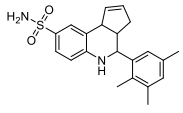
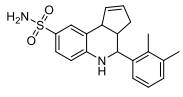
```
            codes[i] = 'NC'+str(i)
            print "Replaced with: %s" % codes[i]

      m.SetProp('_Name',codes[i])
      m.SetProp('smiles',smiles[i])
      m2 = Chem.AddHs(m) #AddH to ensure more plausible
conformations
      AllChem.EmbedMolecule(m2)
      AllChem.UFFOptimizeMolecule(m2) #optimize geom with UFF
forcefield
      embedded_mols.append(m2)
      cann_smiles.append(smiles[i])
      names.append(codes[i])

#print cann_smiles
print len(embedded_mols), len(cann_smiles), len(names)
if len(embedded_mols) == len(cann_smiles) and len(cann_smiles) ==
len(names):
      SDF_basename = smiles_f.split('.')[0]
      w = Chem.SDWriter(SDF_basename+'.sdf')
      for mol in embedded_mols:
            w.write(mol)

      #used_mols = zip(cann_smiles,names)
      used_mols = []
      for i in range(len(cann_smiles)):

      used_mols.append(''.join([cann_smiles[i],'\t',names[i],'\n']
))
      with open(SDF_basename+"used.smi",'w') as used_mol_out:
            used_mol_out.writelines(used_mols)
```

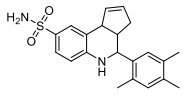
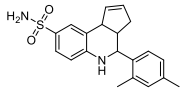
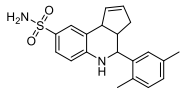
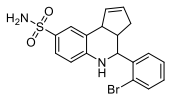
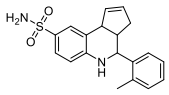
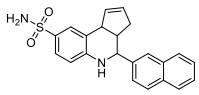
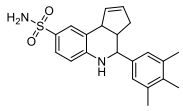
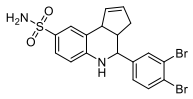
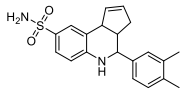
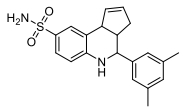
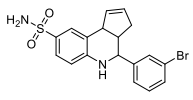
1.1.4 Chapter 3 – Appendix 4: Table of A-867744, TBS and TQS family compounds

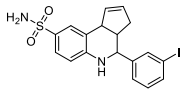
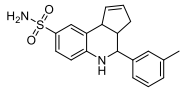
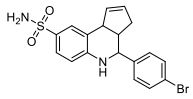
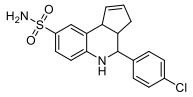
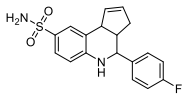
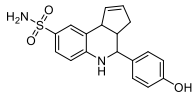
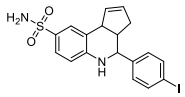
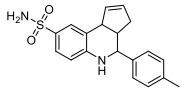
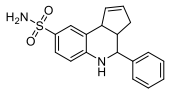
Short Name	Structure	Compound Name	Ref.
		A-867744 Family	
A-867744		4-(5-(4-chlorophenyl)-2-methyl-3-propionyl-1*H*-pyrrol-1-yl)benzenesulfonamide	(2)
A10a		ethyl 5-(4-chlorophenyl)-2-methyl-1-(4-sulfamoylphenyl)-1*H*-pyrrole-3-carboxylate	(2)
A10b		ethyl 5-(4-bromophenyl)-2-methyl-1-(4-sulfamoylphenyl)-1*H*-pyrrole-3-carboxylate	(2)
A15		4-(3-(4-chloro-3-methylbenzoyl)-5-(4-chlorophenyl)-2-methyl-1*H*-pyrrol-1-yl)benzenesulfonamide	(2)
A18		4-(3-acetyl-5-(4-chlorophenyl)-2-methyl-1*H*-pyrrol-1-yl)benzenesulfonamide	(2)
A20		4-(5-(4-chlorophenyl)-3-(cyclopropanecarbonyl)-2-methyl-1*H*-pyrrol-1-yl)benzenesulfonamide	(2)
A21		4-(5-(4-chlorophenyl)-2-methyl-3-(3-methylbutanoyl)-1*H*-pyrrol-1-yl)benzenesulfonamide	(2)
A28		5-(4-chlorophenyl)-*N*-(2-hydroxyethyl)-2-methyl-*N*-propyl-1-(4-sulfamoylphenyl)-1*H*-pyrrole-3-carboxamide	(2)
		TBS-516 Family	

TBS516		4-(5-benzyl-3-(4-bromophenyl)-1*H*-1,2,4-triazol-1-yl)benzenesulfonamide	(3)
TBS546		4-(3-(4-bromophenyl)-5-propyl-1*H*-1,2,4-triazol-1-yl)benzenesulfonamide	(3)
TBS345		4-(3-(4-bromophenyl)-5-phenyl-1*H*-1,2,4-triazol-1-yl)benzenesulfonamide	(3)
TBS556		4-(3-(4-bromophenyl)-5-phenethyl-1*H*-1,2,4-triazol-1-yl)benzenesulfonamide	(3)

TQS-Family

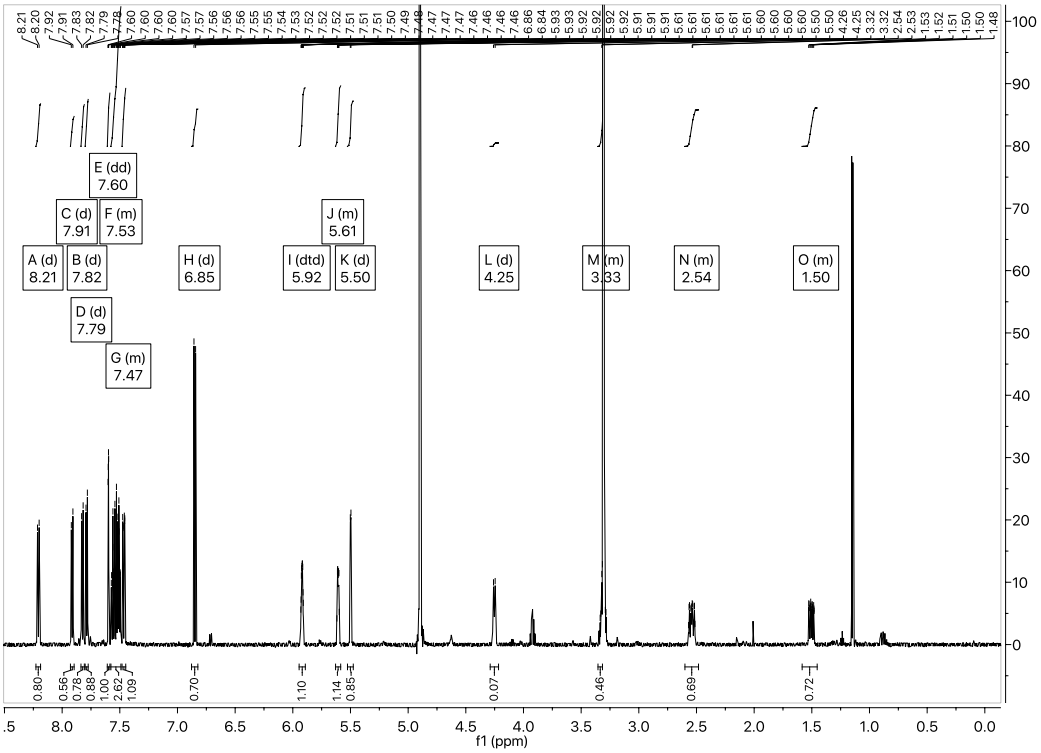
TQS		*cis*-*cis*-4-(napthalen-1-yl)-3*a*,4,5,9*b*-tetrahydro-3*H*-cyclopenta[*c*]quinoline-8-sulfonamide	(4)
2345MP-TQS		*cis*-*cis*-4-(2,3,4,5-tetramethylphenyl)-3a,4,5,9b-tetrahydro-3*H*-cyclopenta[*c*]quinoline-8-sulfonamide	(5)
234MP-TQS		*cis*-*cis*-4-(2,3,4-trimethylphenyl)-3a,4,5,9b-tetrahydro-3*H*-cyclopenta[*c*]quinoline-8-sulfonamide	(5)
235MP-TQS		*cis*-*cis*-4-(2,3-dimethylphenyl)-3a,4,5,9b-tetrahydro-3*H*-cyclopenta[*c*]quinoline-8-sulfonamide	(5)
23MP-TQS		*cis*-*cis*-4-(2,3-dimethylphenyl)-3a,4,5,9b-tetrahydro-3*H*-cyclopenta[*c*]quinoline-8-sulfonamide	(5)

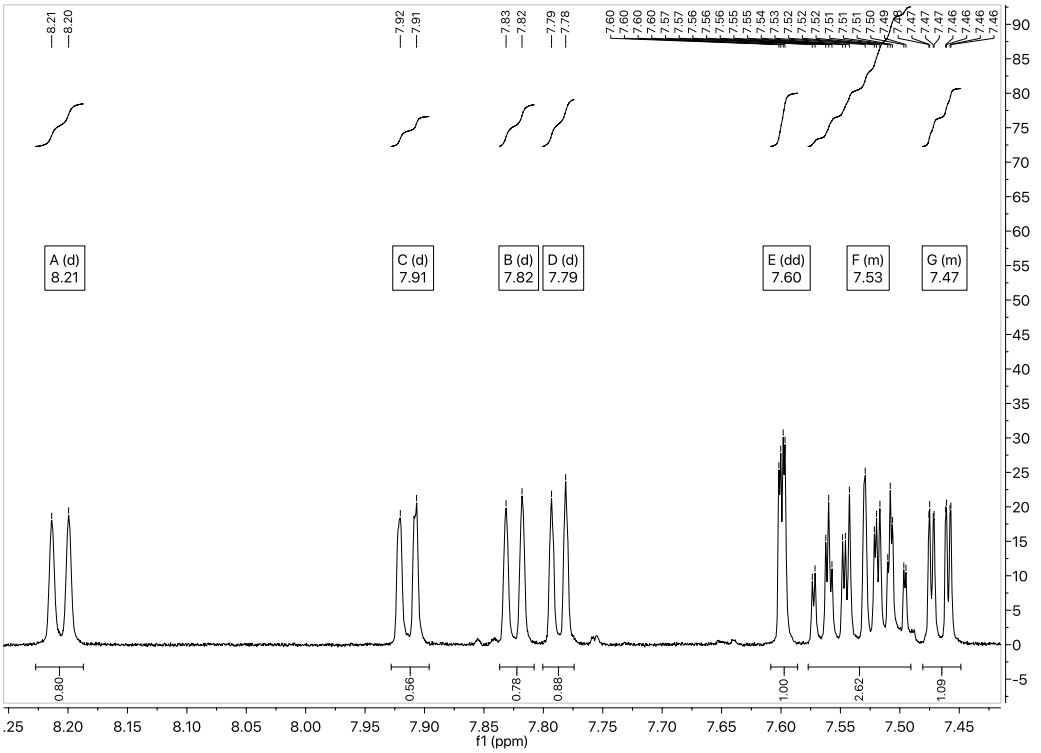
245MP-TQS		*cis-cis*-4-(2,4,5-trimethylphenyl)-3a,4,5,9b-tetrahydro-3*H*-cyclopenta[*c*]quinoline-8-sulfonamide	(5)
24MP-TQS		*cis-cis*-4-(2,4-dimethylphenyl)-3a,4,5,9b-tetrahydro-3*H*-cyclopenta[*c*]quinoline-8-sulfonamide	(5)
25MP-TQS		*cis-cis*-4-(2,5-dimethylphenyl)-3a,4,5,9b-tetrahydro-3*H*-cyclopenta[*c*]quinoline-8-sulfonamide	(5)
2BP-TQS		*cis-cis*-4-(2-bromophenyl)-3a,4,5,9b-tetrahydro-3*H*-cyclopenta[*c*]quinoline-8-sulfonamide	(6)
2MP-TQS		*cis-cis*-4-(o-tolyl)-3a,4,5,9b-tetrahydro-3*H*-cyclopenta[*c*]quinoline-8-sulfonamide	(5)
2N-TQS		*cis-cis*-4-(naphthalen-2-yl)-3a,4,5,9b-tetrahydro-3*H*-cyclopenta[*c*]quinoline-8-sulfonamide	(6)
345MP-TQS		*cis-cis*-4-(3,4,5-trimethylphenyl)-3a,4,5,9b-tetrahydro-3*H*-cyclopenta[*c*]quinoline-8-sulfonamide	(5)
34BP-TQS		*cis-cis*-4-(3,4-dibromophenyl)-3a,4,5,9b-tetrahydro-3*H*-cyclopenta[*c*]quinoline-8-sulfonamide	(6)
34MP-TQS		*cis-cis*-4-(3,4-dimethylphenyl)-3a,4,5,9b-tetrahydro-3*H*-cyclopenta[*c*]quinoline-8-sulfonamide	(5)
35MP-TQS		*cis-cis*-4-(3,5-dimethylphenyl)-3a,4,5,9b-tetrahydro-3*H*-cyclopenta[*c*]quinoline-8-sulfonamide	(5)
3BP-TQS		*cis-cis*-4-(3-bromophenyl)-3a,4,5,9b-tetrahydro-3*H*-cyclopenta[*c*]quinoline-8-sulfonamide	(6)

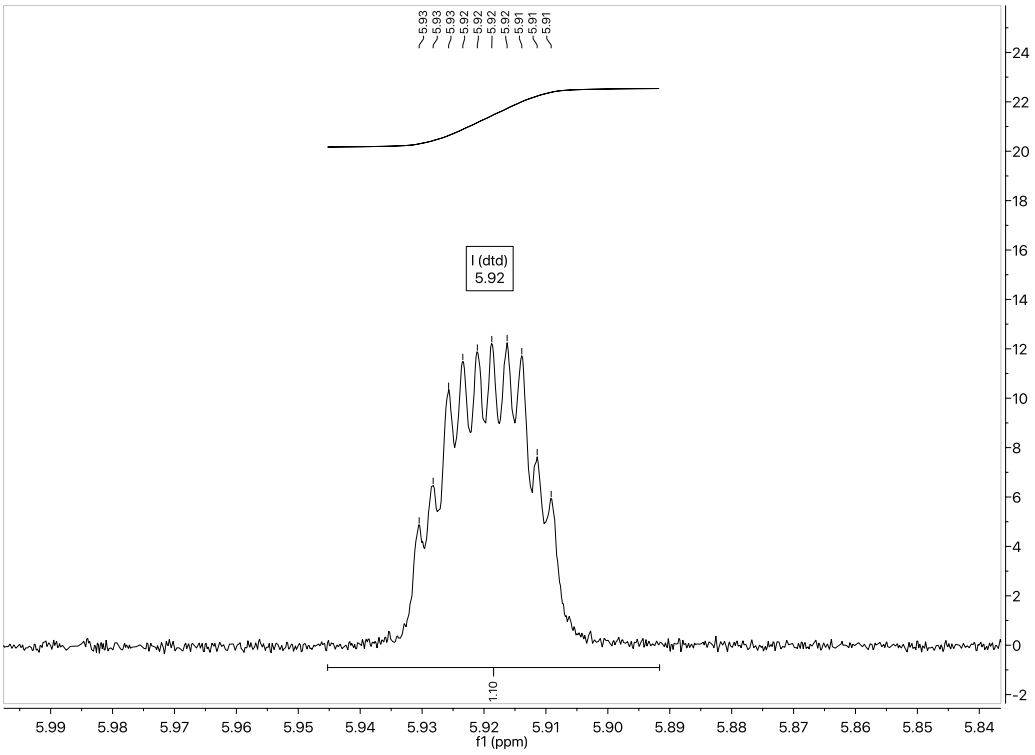
3IP-TQS		*cis-cis*-4-(3-iodophenyl)-3a,4,5,9b-tetrahydro-3*H*-cyclopenta[*c*]quinoline-8-sulfonamide	(6)
3MP-TQS		*cis-cis*-4-(m-tolyl)-3a,4,5,9b-tetrahydro-3*H*-cyclopenta[*c*]quinoline-8-sulfonamide	(5)
4BP-TQS		*cis-cis*-4-(4-bromophenyl)-3a,4,5,9b-tetrahydro-3*H*-cyclopenta[*c*]quinoline-8-sulfonamide	(7)
4CP-TQS		*cis-cis*-4-(4-chlorophenyl)-3a,4,5,9b-tetrahydro-3*H*-cyclopenta[*c*]quinoline-8-sulfonamide	(6)
4FP-TQS		*cis-cis*-4-(4-fluorophenyl)-3a,4,5,9b-tetrahydro-3*H*-cyclopenta[*c*]quinoline-8-sulfonamide	(6)
4HP-TQS		*cis-cis*-4-(4-hydroxyphenyl)-3a,4,5,9b-tetrahydro-3*H*-cyclopenta[*c*]quinoline-8-sulfonamide	(6)
4IP-TQS		*cis-cis*-4-(4-iodophenyl)-3a,4,5,9b-tetrahydro-3*H*-cyclopenta[*c*]quinoline-8-sulfonamide	(6)
4MP-TQS		*cis-cis*-4-(p-tolyl)-3a,4,5,9b-tetrahydro-3*H*-cyclopenta[*c*]quinoline-8-sulfonamide	(6)
P-TQS		*cis-cis*-4-phenyl-3a,4,5,9b-tetrahydro-3*H*-cyclopenta[*c*]quinoline-8-sulfonamide	(6)

1.1.5 Chapter 3 – Appendix 5 – Chemical data

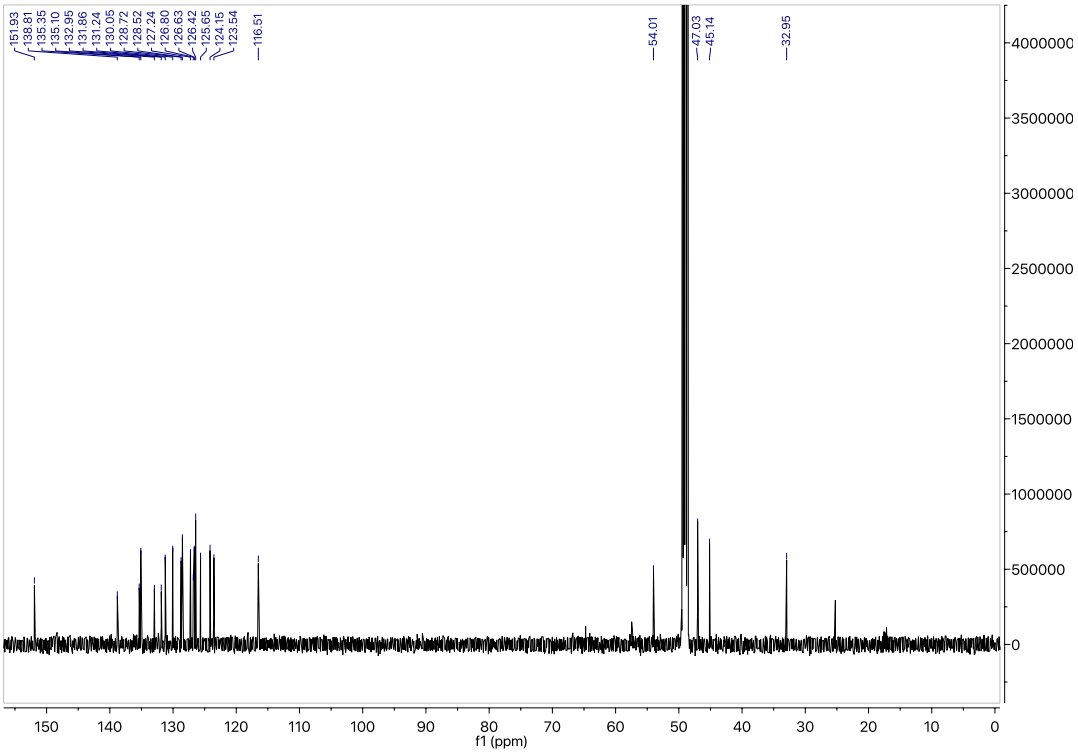
1.1.5.1 TQS ^{1}H NMR



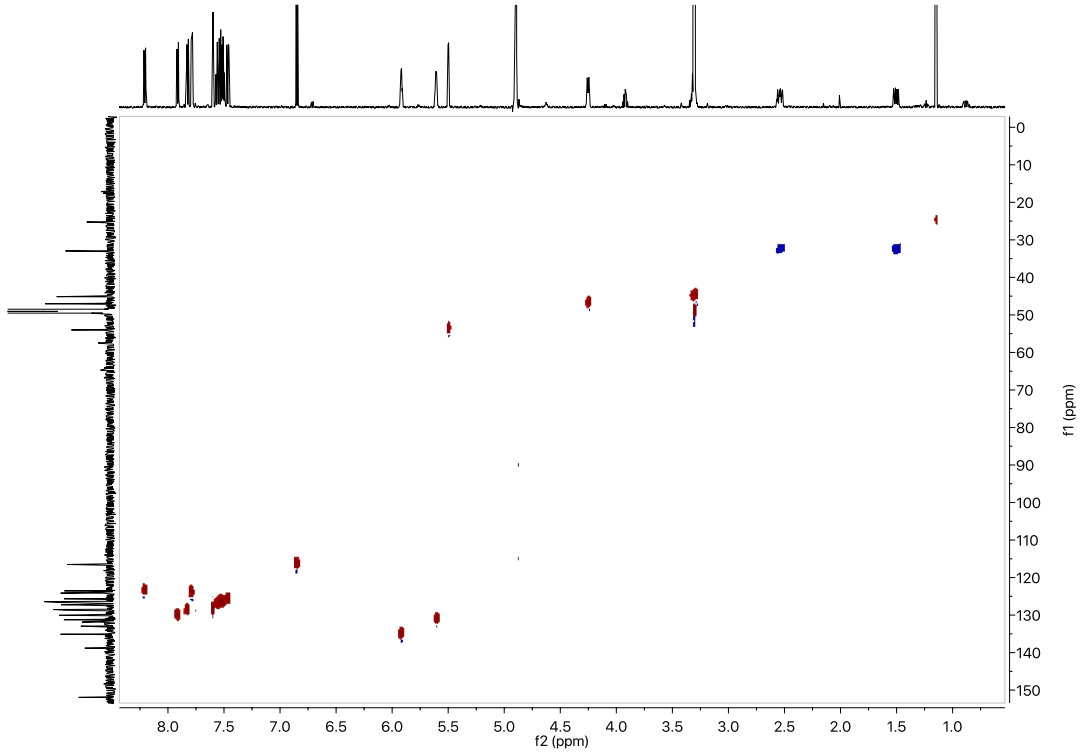




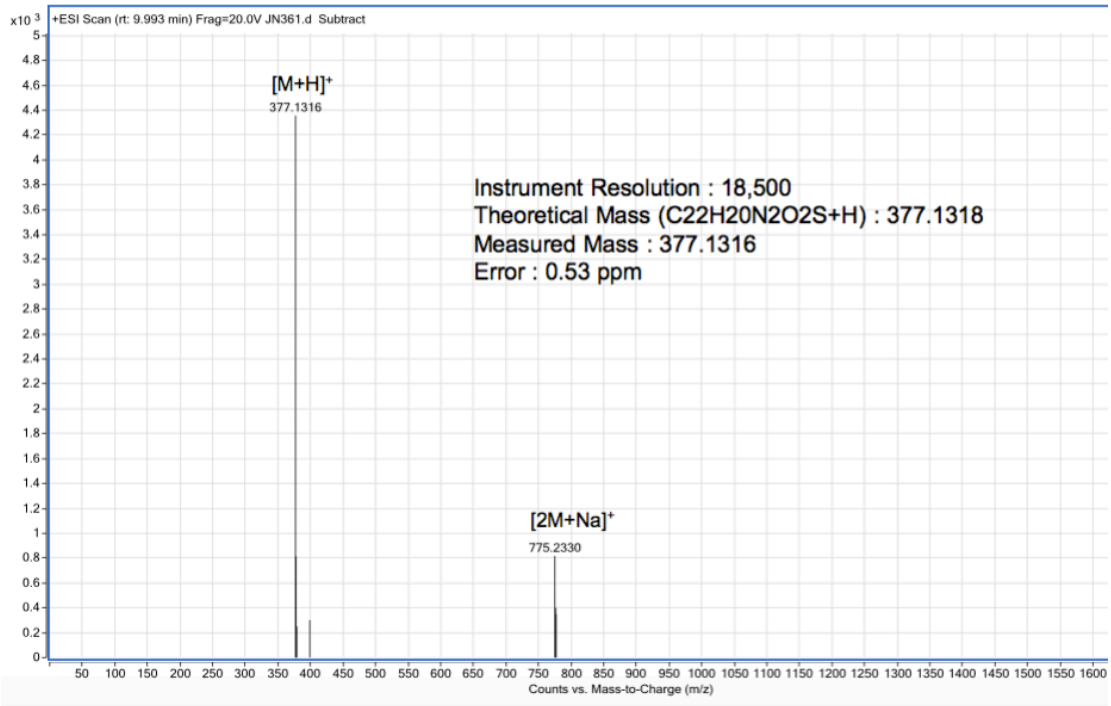
1.1.5.2 TQS ^{13}C NMR



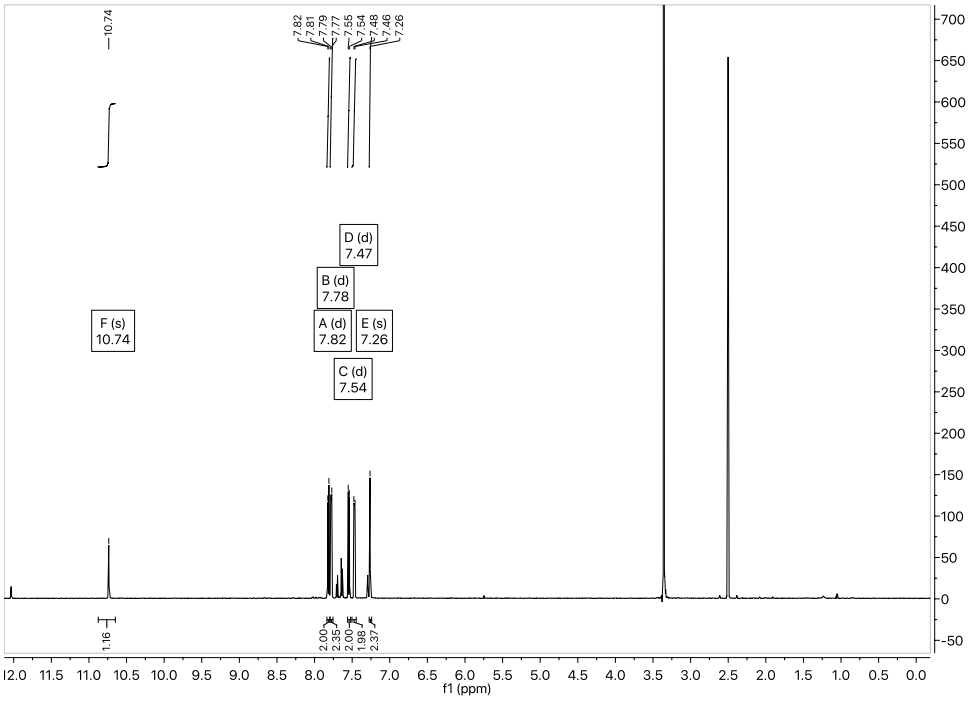
1.1.5.3 TQS HSQC



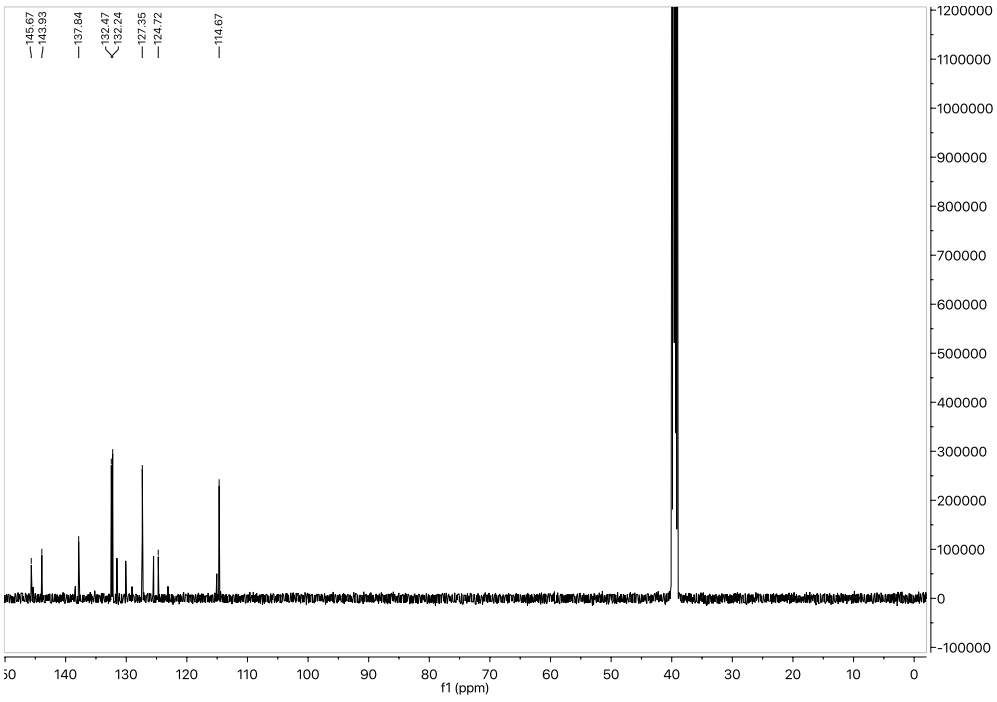
1.1.5.4 TQS Mass Spec



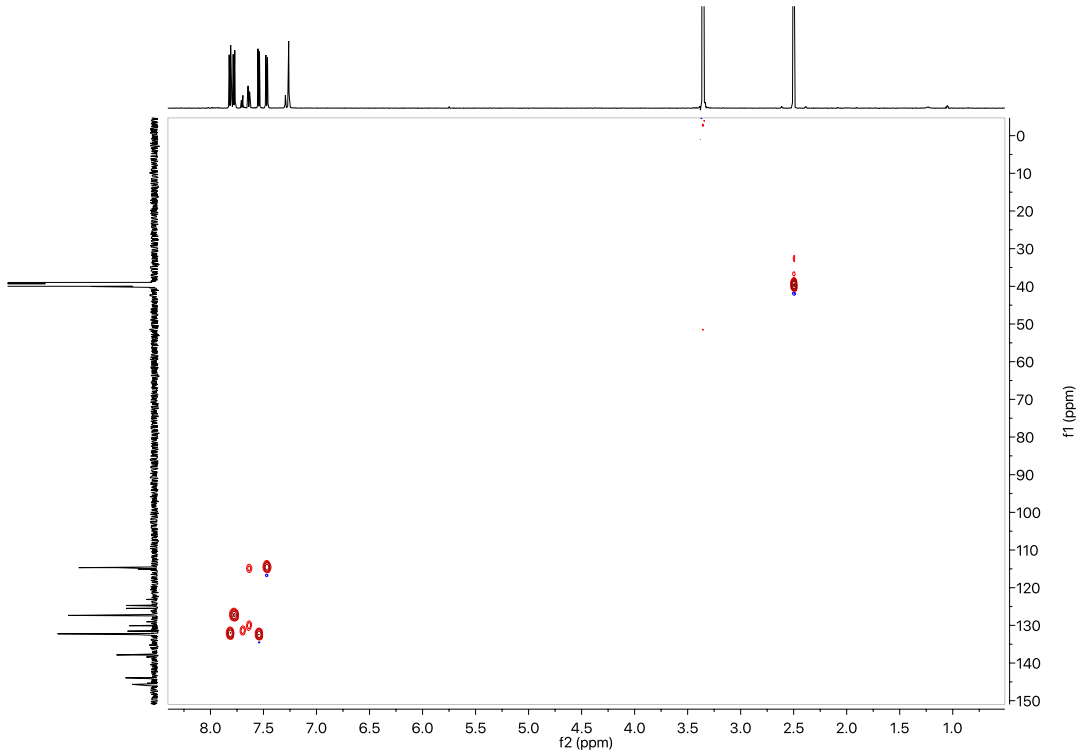
1.1.5.5 4-(2-((4-bromophenyl)(nitro)methylene)hydrazineyl)benzenesulfonamide ^{1}H NMR



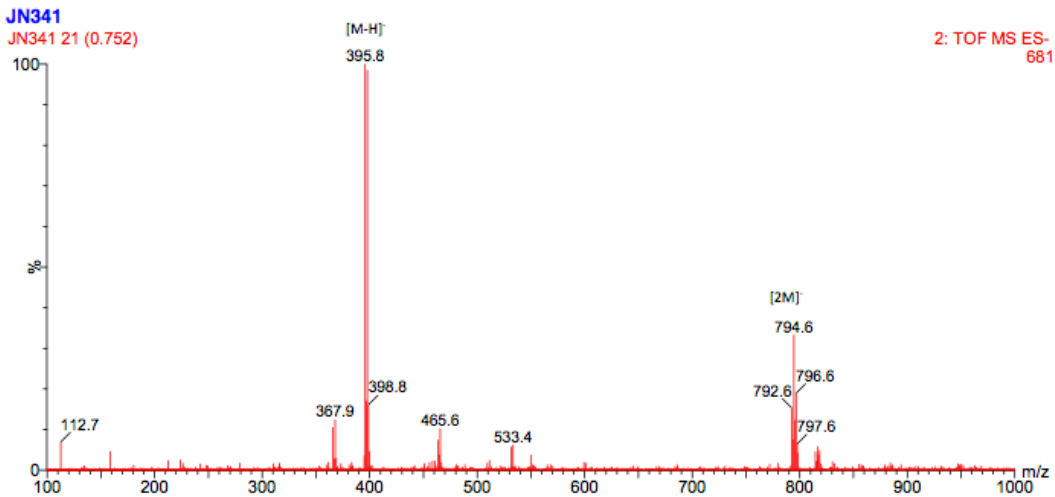
1.1.5.6 4-(2-((4-bromophenyl)(nitro)methylene)hydrazineyl)benzenesulfonamide ^{13}C NMR



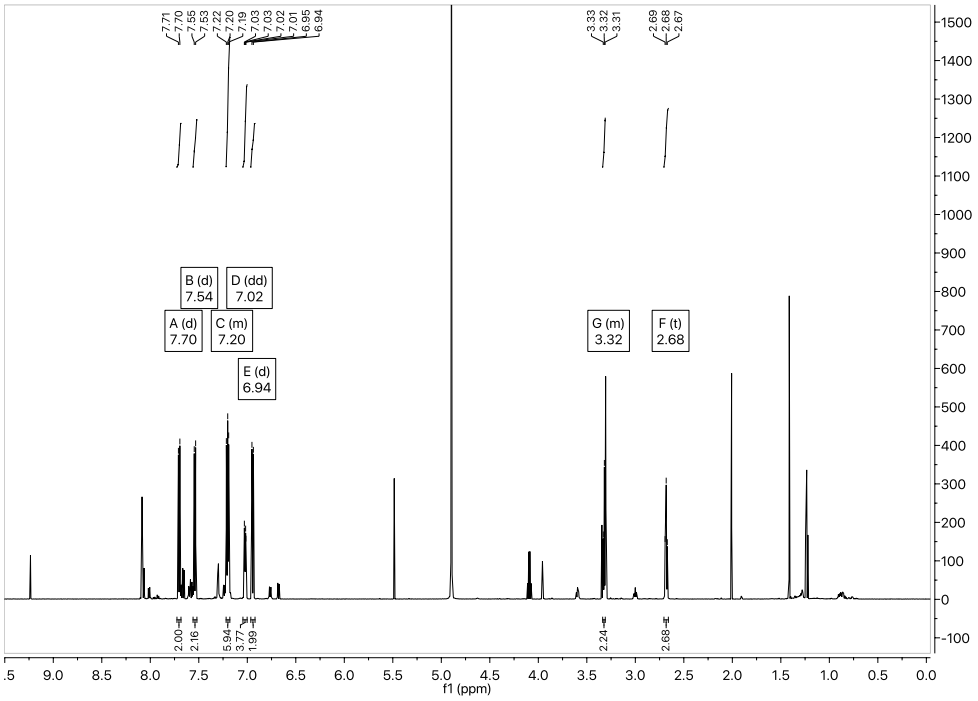
1.1.5.7 4-(2-((4-bromophenyl)(nitro)methylene)hydrazineyl)benzenesulfonamide HSQC



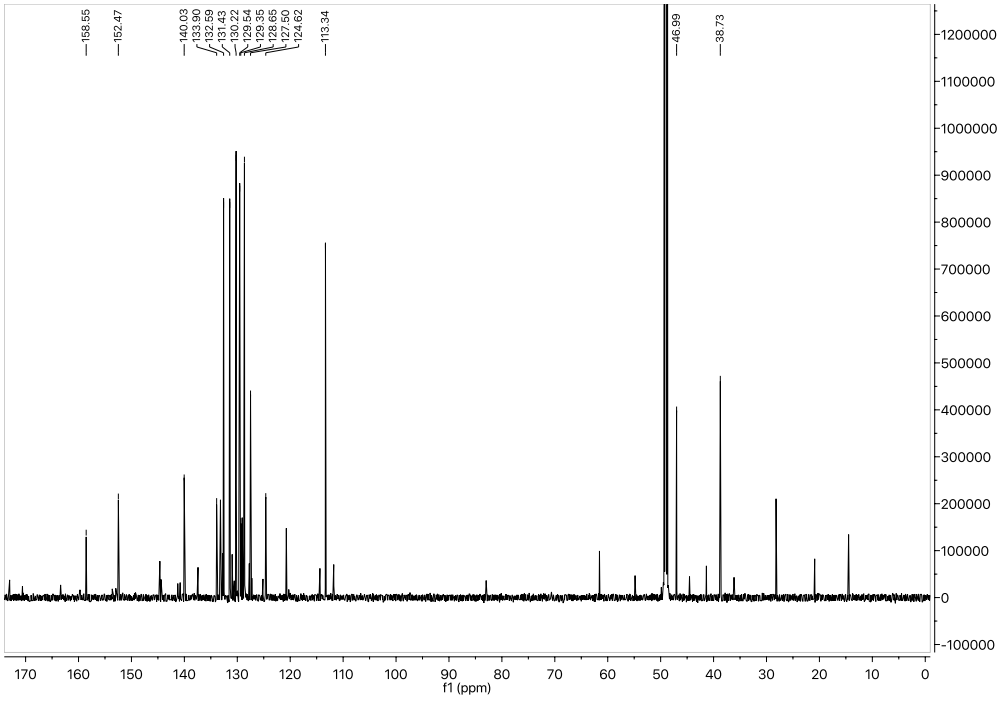
1.1.5.8 4-(2-((4-bromophenyl)(nitro)methylene)hydrazineyl)benzenesulfonamide Mass Spec



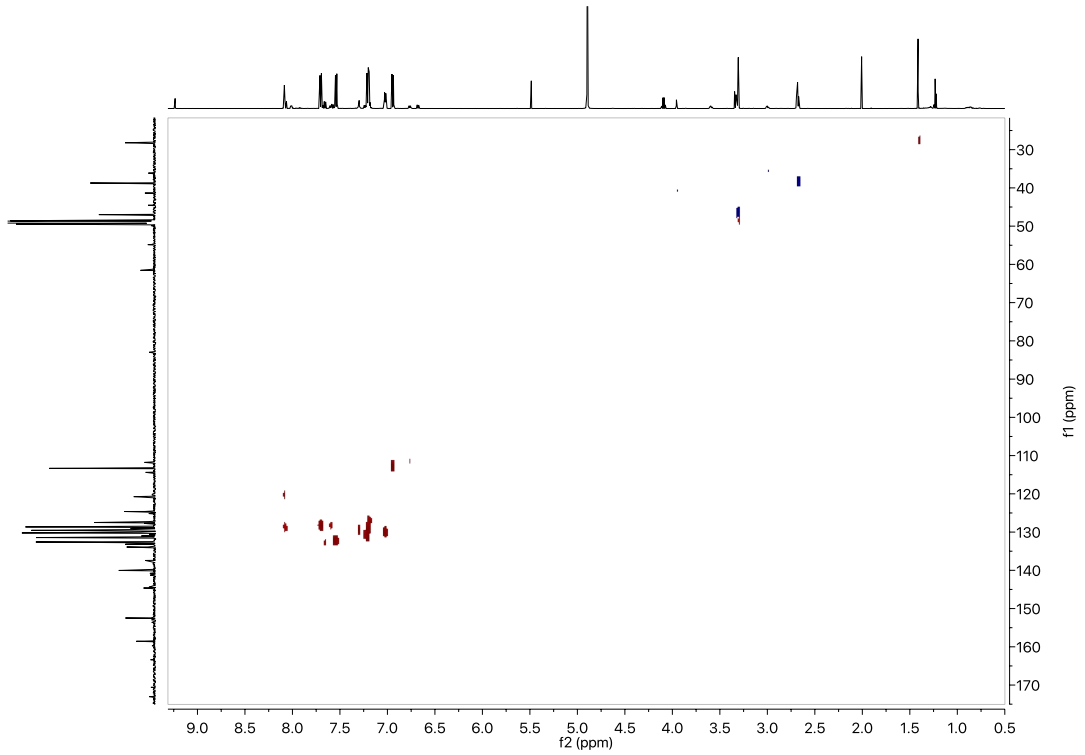
1.1.5.9 4-bromo-N-phenethyl-N'-(4-sulfamoylphenyl)benzohydrazonamide ^{1}H NMR



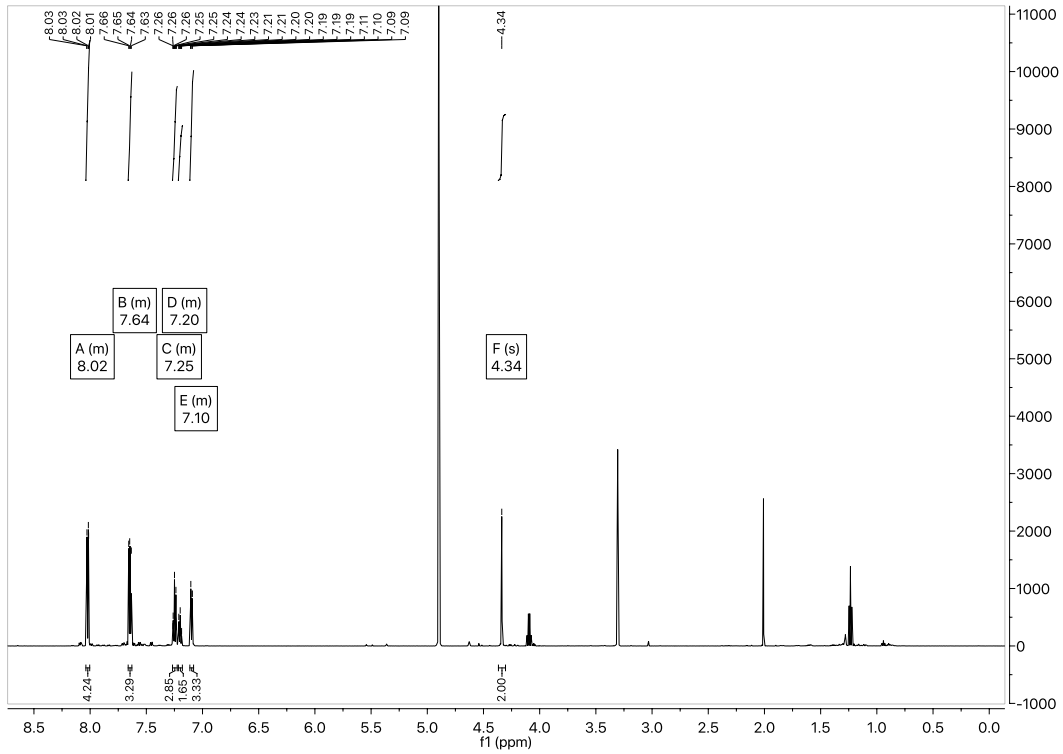
1.1.5.10 4-bromo-N-phenethyl-N'-(4-sulfamoylphenyl)benzohydrazonamide ^{13}C NMR



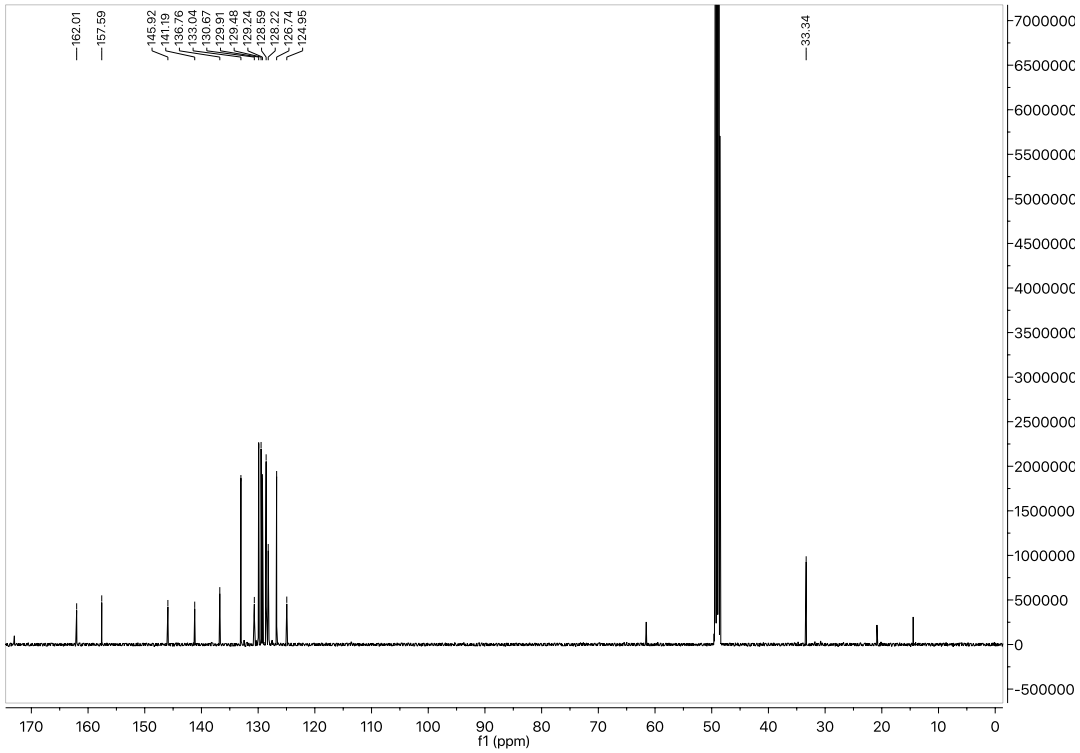
1.1.5.11 4-bromo-N-phenethyl-N'-(4-sulfamoylphenyl)benzohydrazonamide HSQC



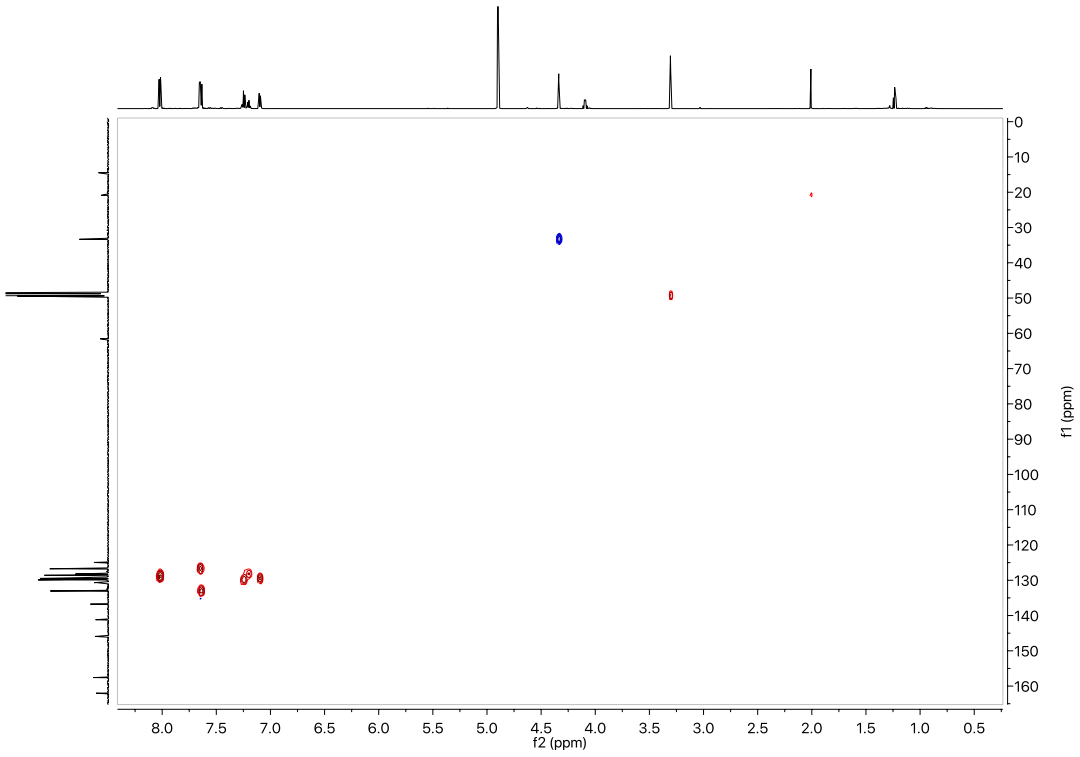
1.1.5.12 TBS-516 ^{1}H NMR



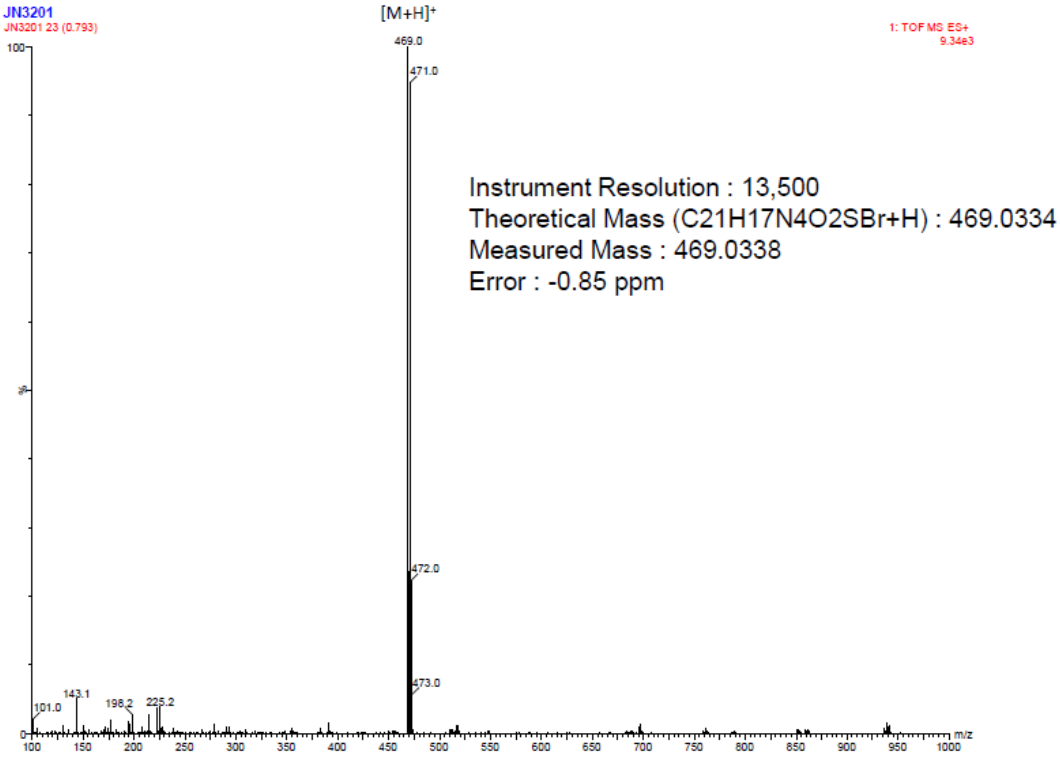
1.1.5.13 TBS-516 ^{13}C NMR



1.1.5.14 TBS-516 HSQC



1.1.5.15 TBS-516 Mass Spec

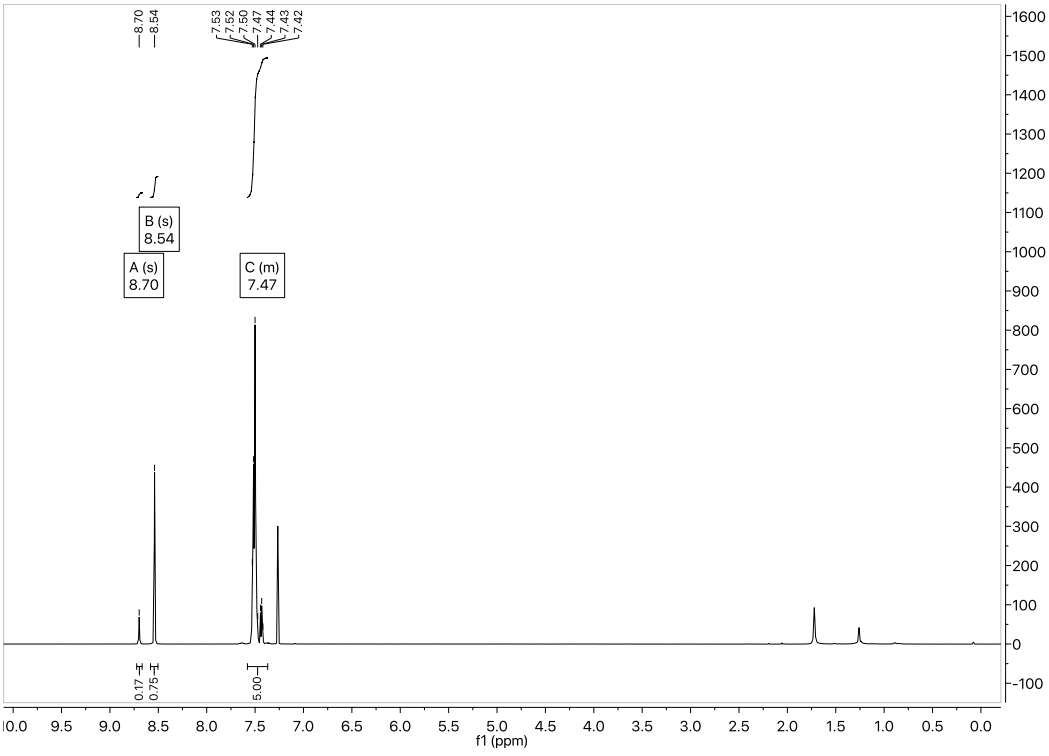


1.2 Appendix – Chapter 4

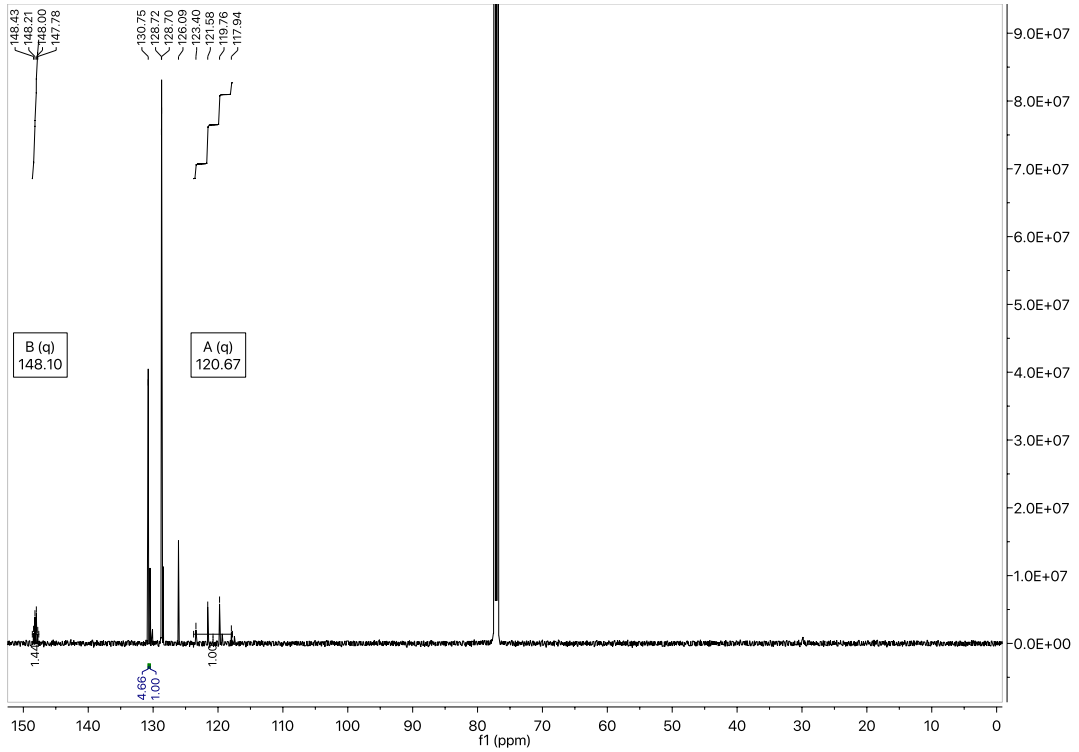
1.2.1 Chapter 4 – Appendix 1: Chemical data

1.2.1.1 2,2,2-Trifluoro-1-phenylethan-1-one oxime

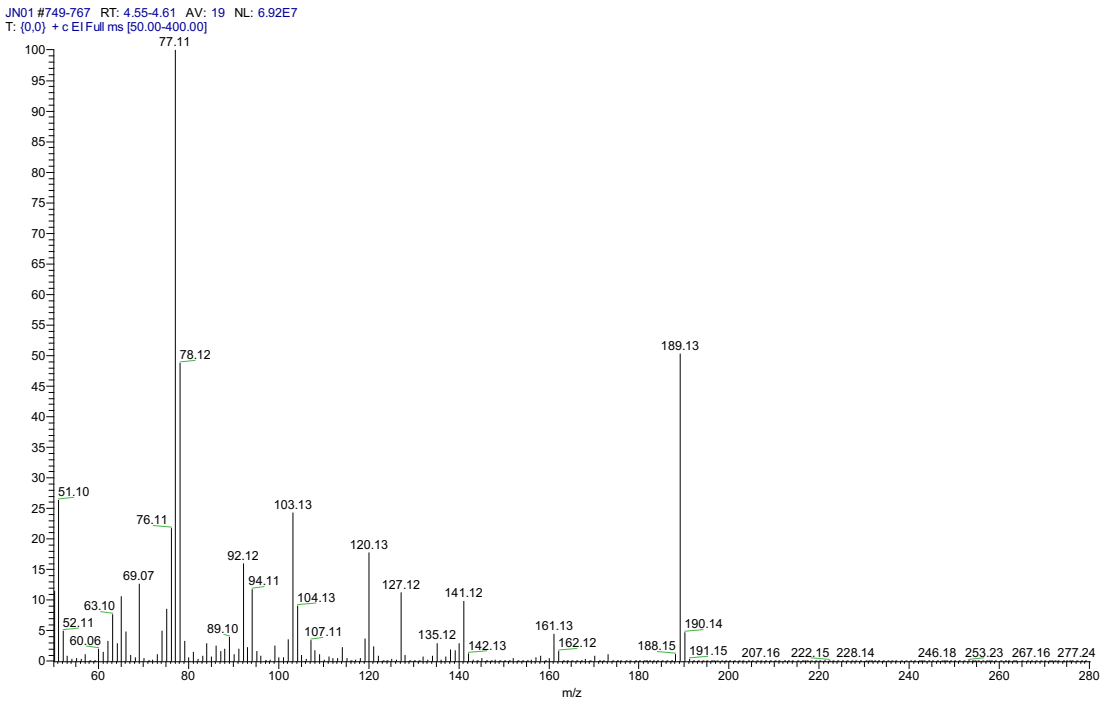
1.2.1.1.1 ^{1}H NMR



1.2.1.1.2 ^{13}C NMR

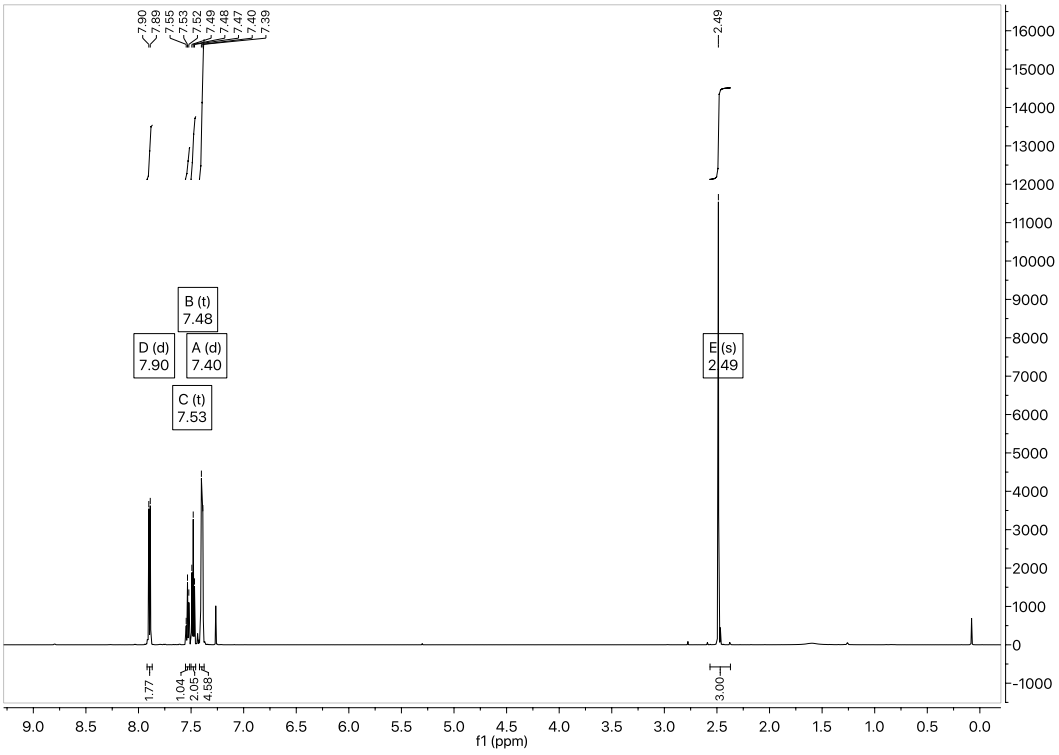


1.2.1.1.3 LRMS

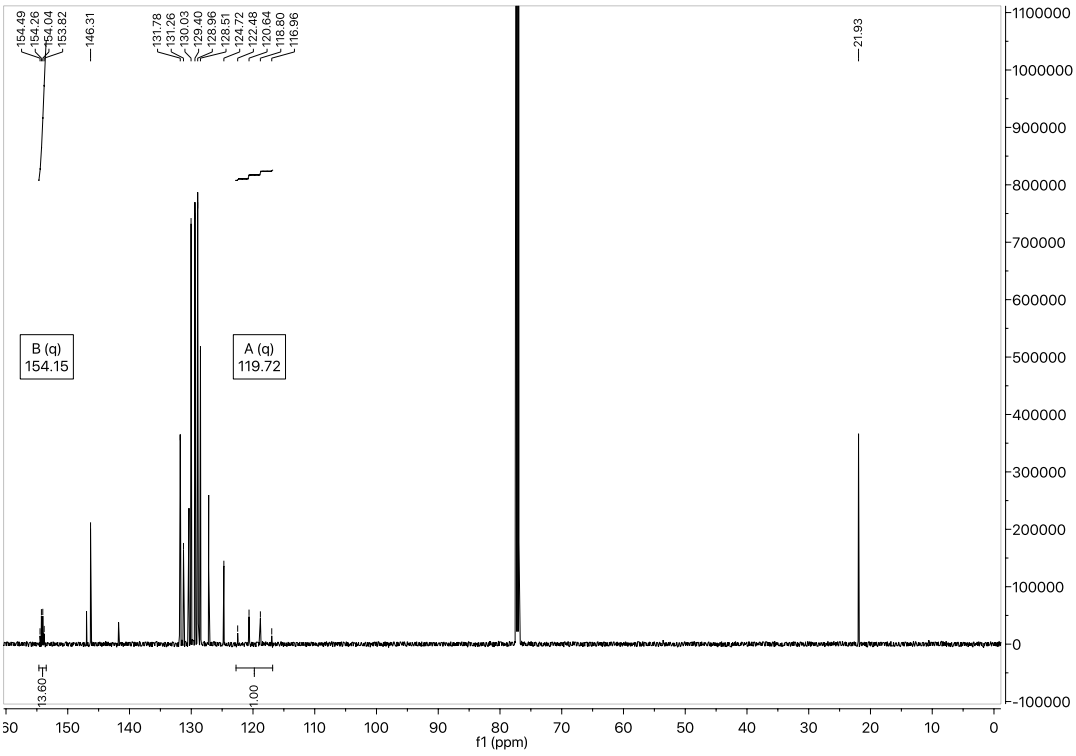


1.2.1.2 2,2,2-Trifluoro-1-phenylethan-1-one O-tosyl oxime

1.2.1.2.1 ^{1}H NMR

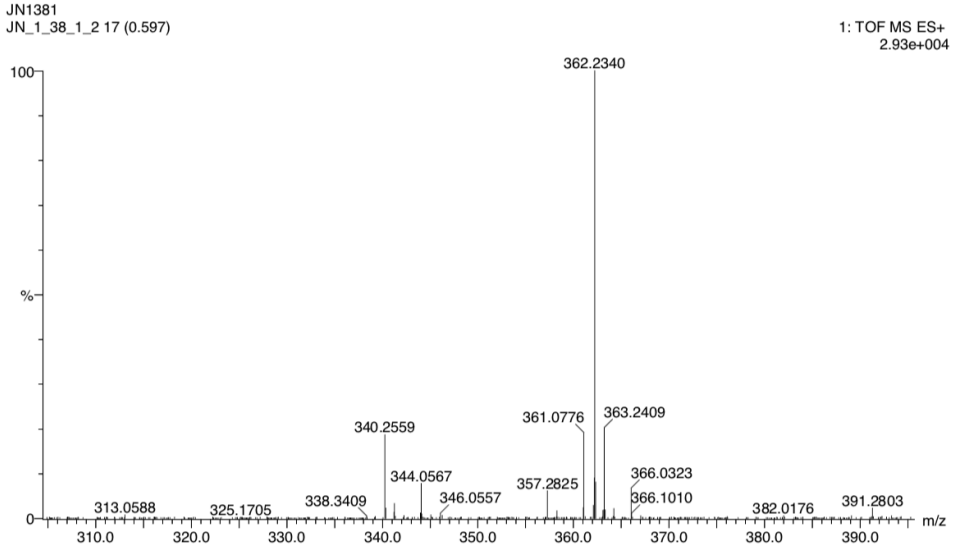


1.2.1.2.2 ^{13}C NMR



1.2.1.2.3 HRMS

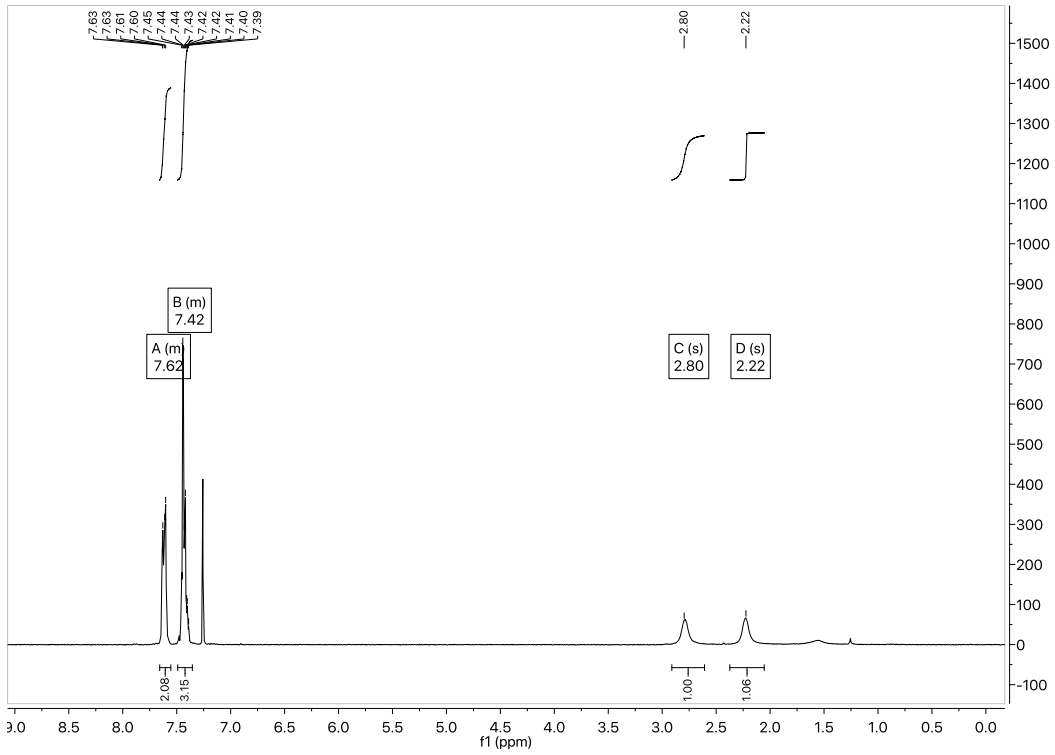
Monoisotopic Mass, Even Electron Ions
164 formula(e) evaluated with 1 results within limits (all results (up to 1000) for each mass)
Elements Used:
C: 0-15 H: 0-15 N: 0-3 O: 0-3 F: 0-3 S: 0-2
JN1381
JN_1_38_1_2 17 (0.597)



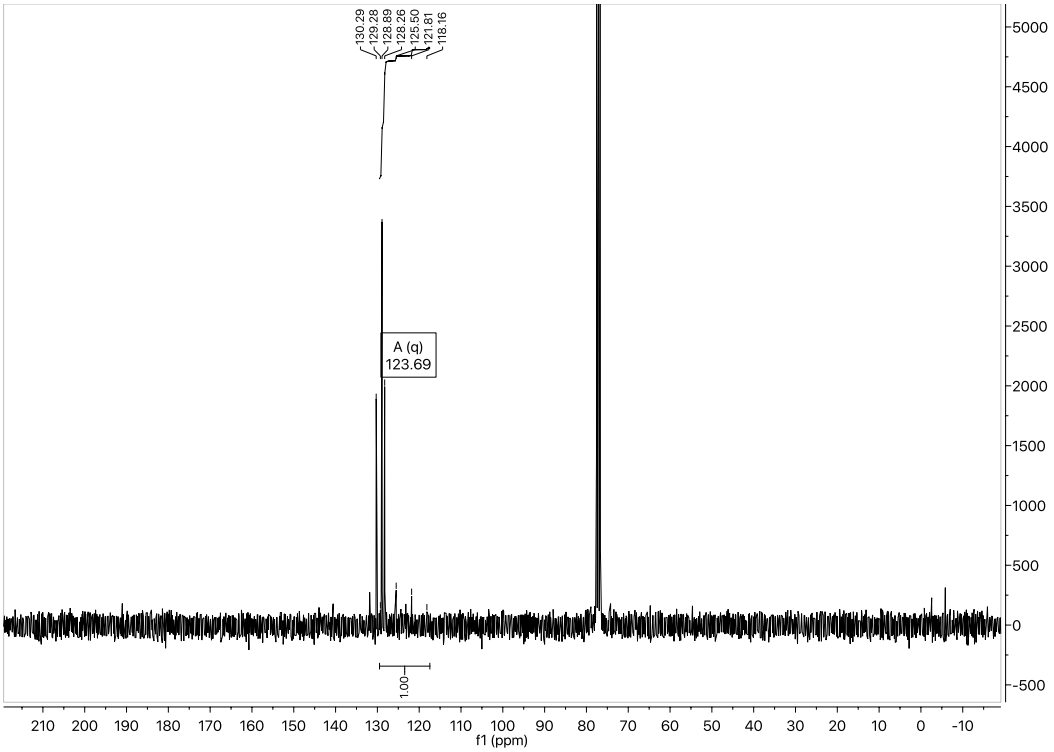
Minimum:				-1.5		
Maximum:		5.0	5.0	50.0		
Mass	Calc. Mass	mDa	PPM	DBE	i-FIT	Formula
344.0567	344.0568	-0.1	-0.3	8.5	122.1	C15 H13 N O3 F3 S

1.2.1.3 3-Phenyl-3-(trifluoromethyl)diaziridine

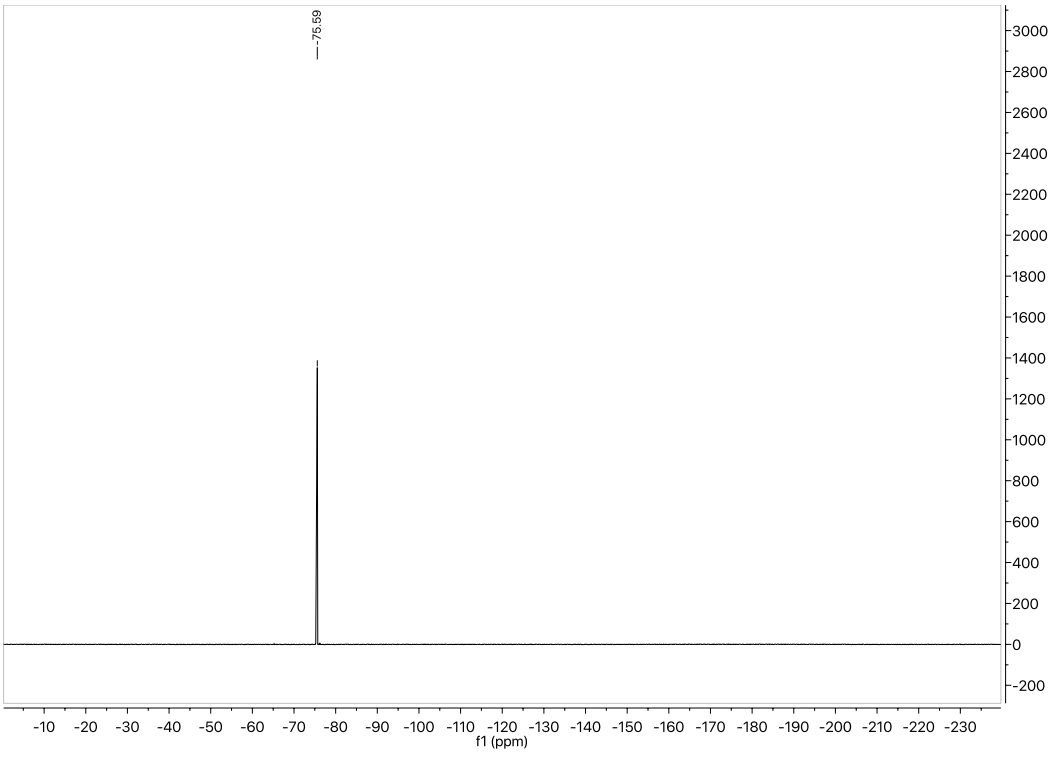
1.2.1.3.1 ^{1}H NMR



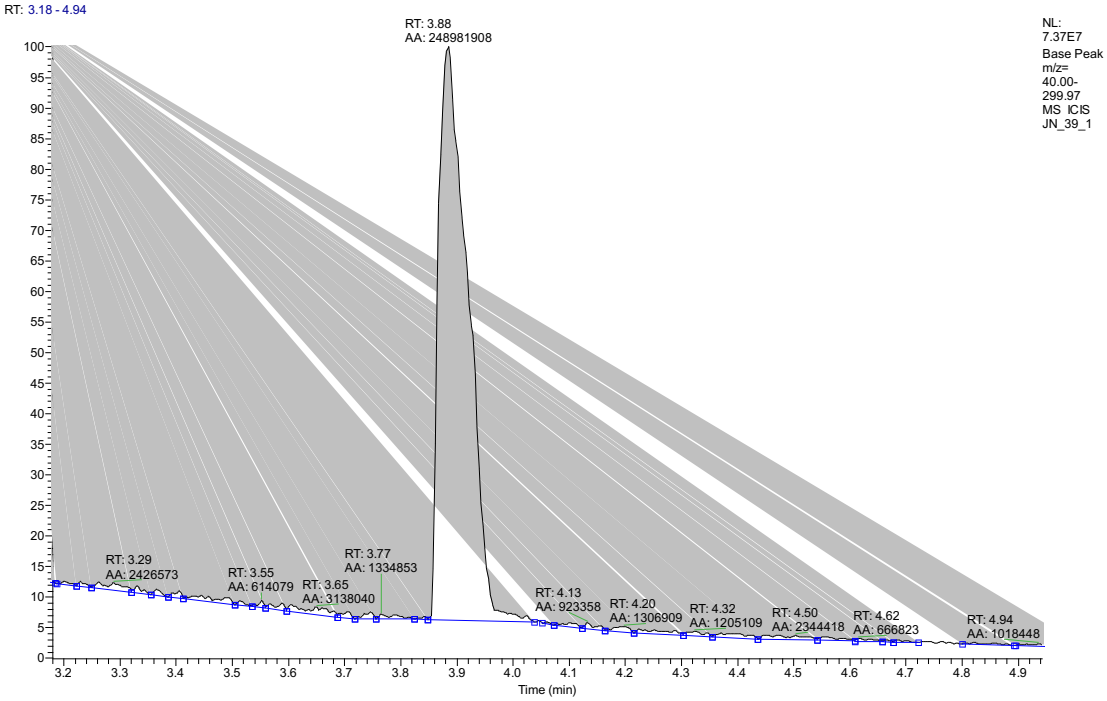
1.2.1.3.2 ^{13}C NMR

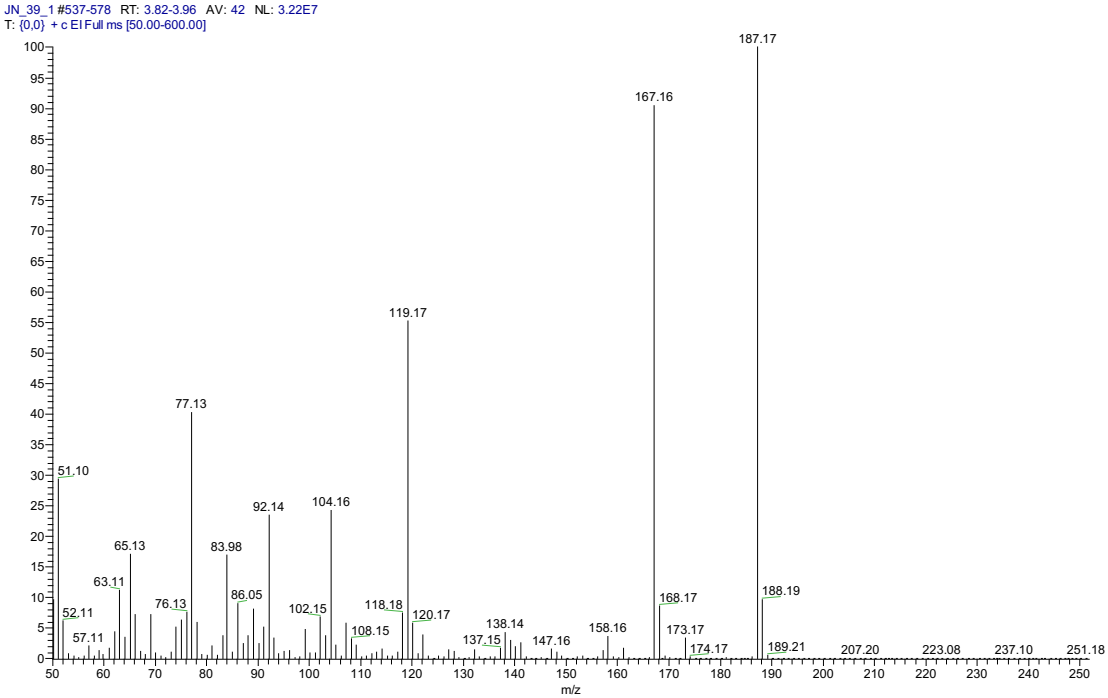


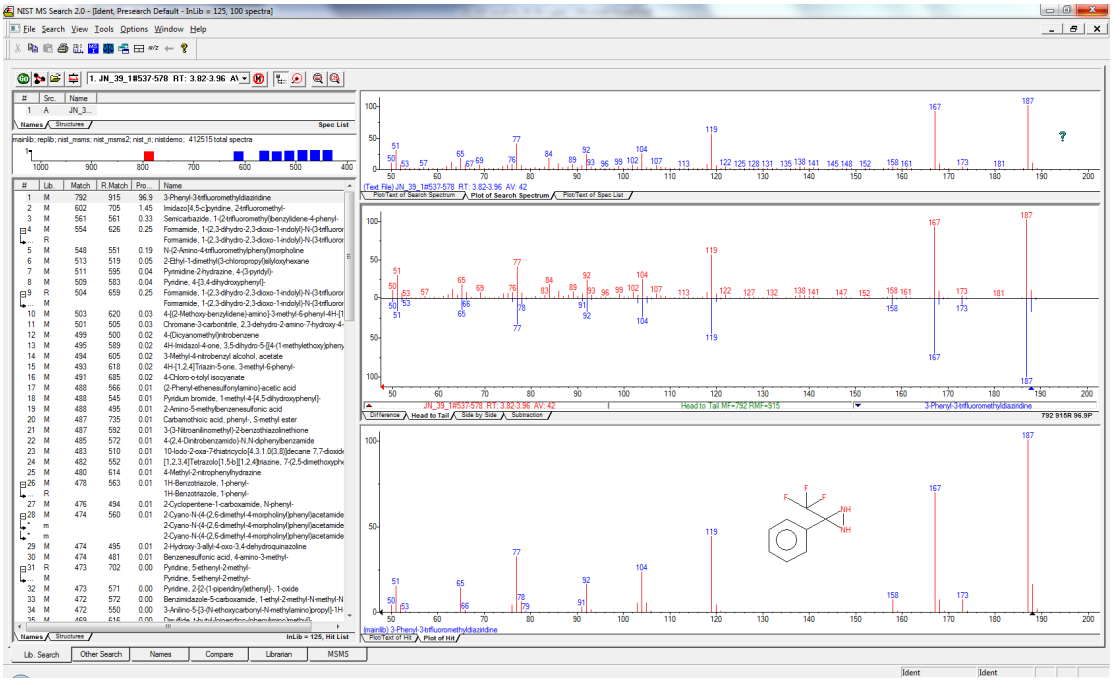
1.2.1.3.3 ^{19}F NMR



1.2.1.3.4 GCMS

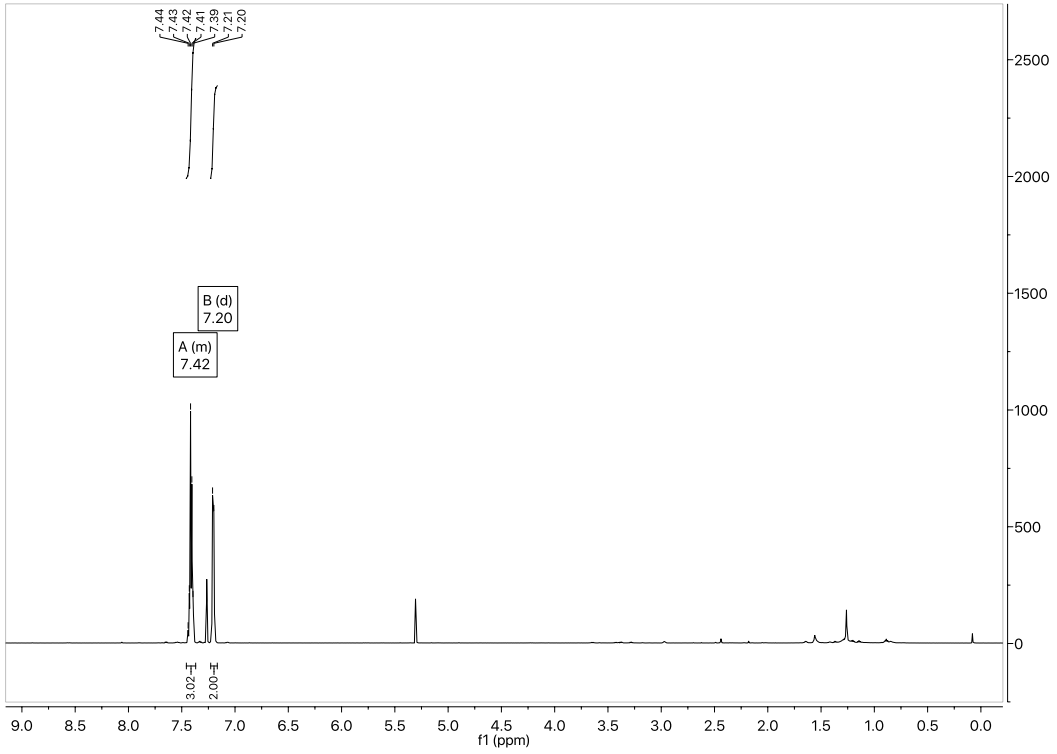

RT: 3.18 - 4.94
RT: 3.88
AA: 248981908
NL:
7.37E7
Base Peak
m/z=
40.00-
299.97
MS ICIS
JN_39_1
RT: 3.29
AA: 2426573
RT: 3.55
AA: 614079
RT: 3.65
AA: 3138040
RT: 3.77
AA: 1334853
RT: 4.13
AA: 923358
RT: 4.20
AA: 1306909
RT: 4.32
AA: 1205109
RT: 4.50
AA: 2344418
RT: 4.62
AA: 666823
RT: 4.94
AA: 1018448
Time (min)


JN_39_1 #537-578 RT: 3.82-3.96 AV: 42 NL: 3.22E7
T: {0,0} + c EI Full ms [50.00-600.00]
187.17
167.16
119.17
77.13
51.10
92.14
104.16
65.13
83.98
63.11
86.05
76.13
102.15
118.18
120.17
52.11
57.11
108.15
138.14
137.15
147.16
158.16
168.17
173.17
174.17
188.19
189.21
207.20
223.08
237.10
251.18
m/z

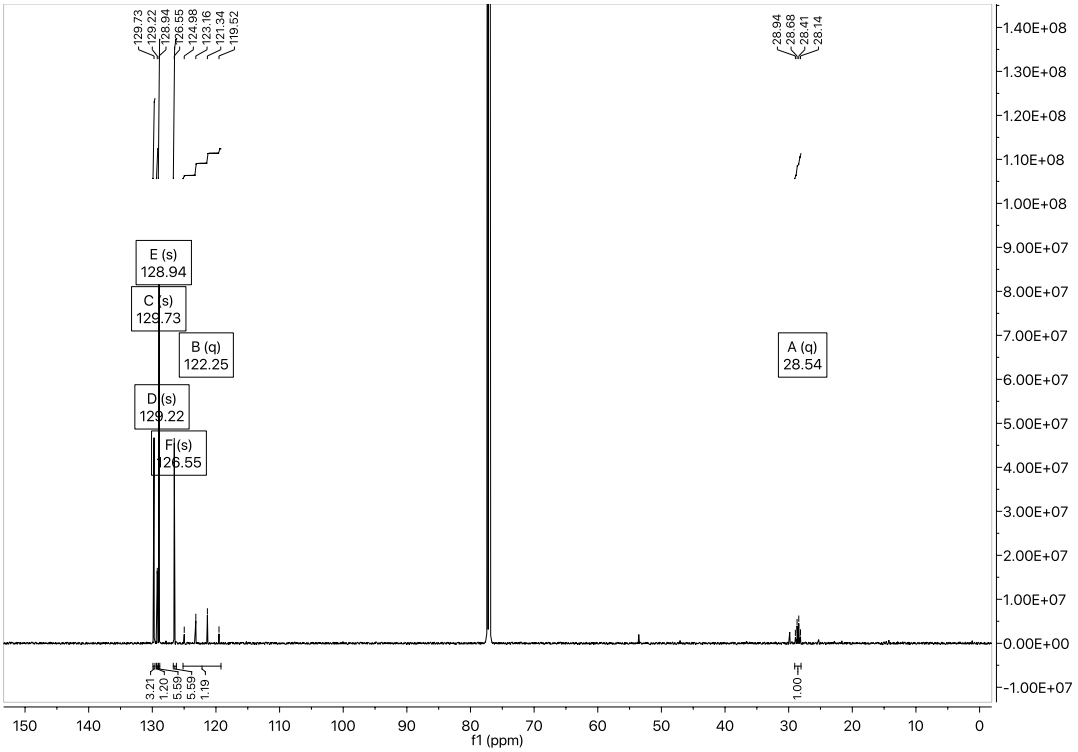


1.2.1.4 3-Phenyl-3-(trifluoromethyl)-3H-diazirine

1.2.1.4.1 ^{1}H NMR

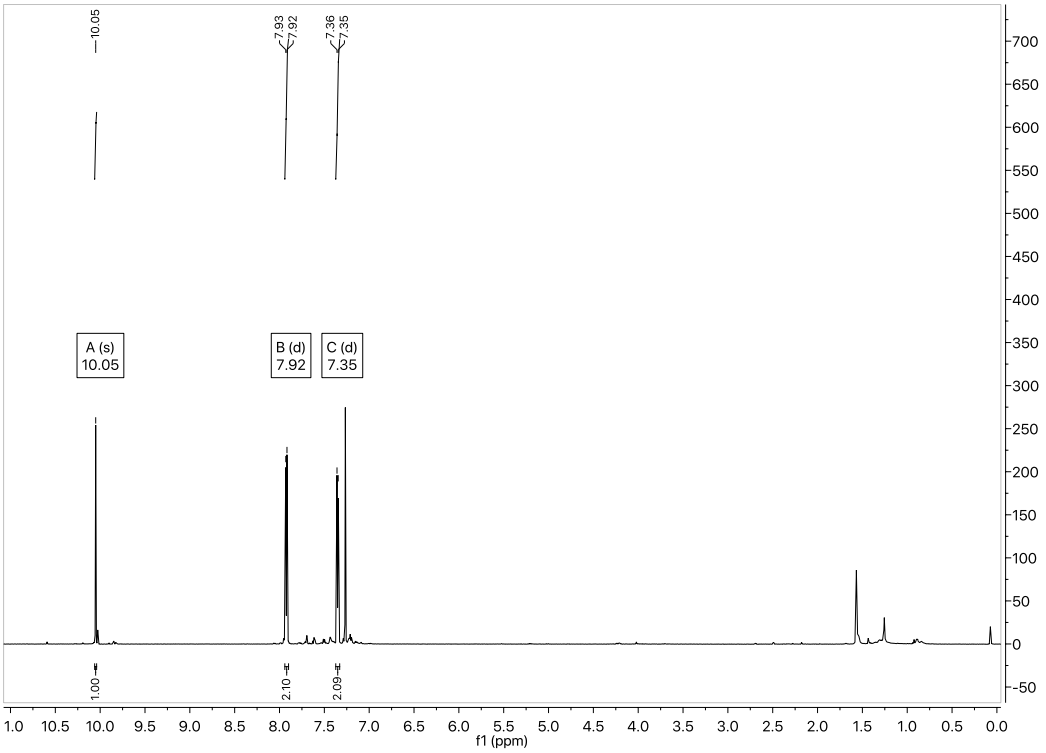


1.2.1.4.2 ^{13}C NMR

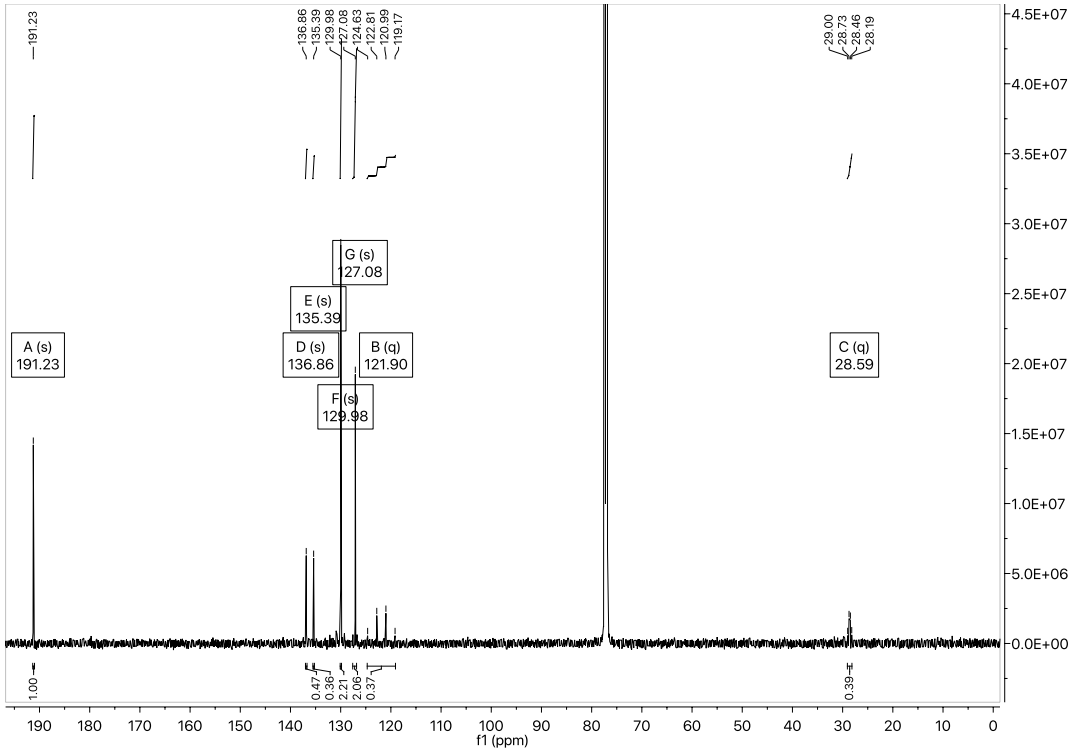


1.2.1.5 4-(3-(Trifluoromethyl)-3H-diazirin-3-yl)benzaldehyde

1.2.1.5.1 ^{1}H NMR

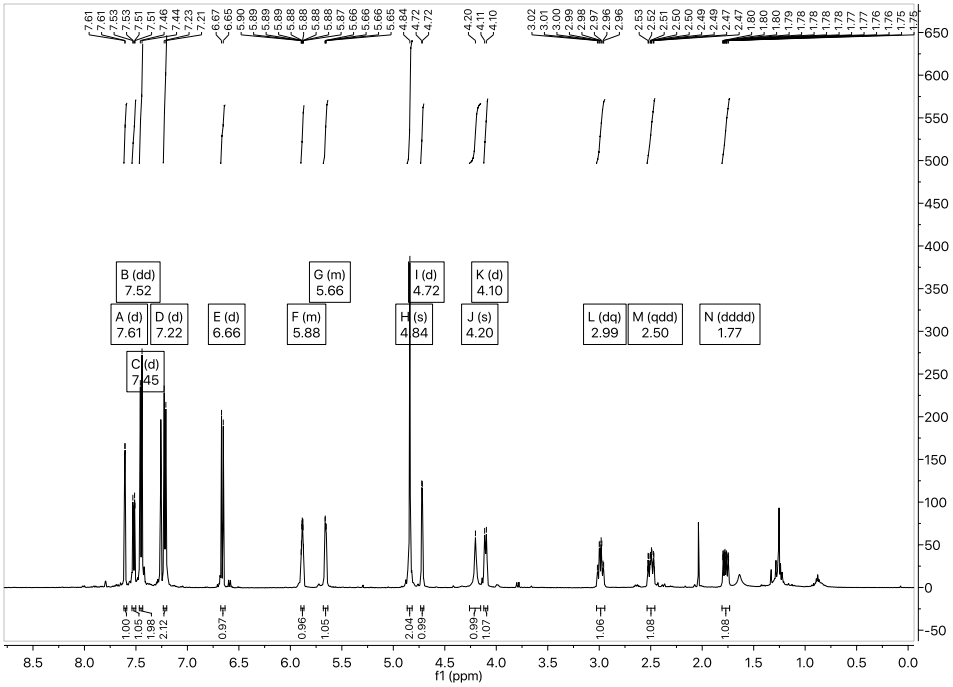


1.2.1.5.2 ^{13}C NMR

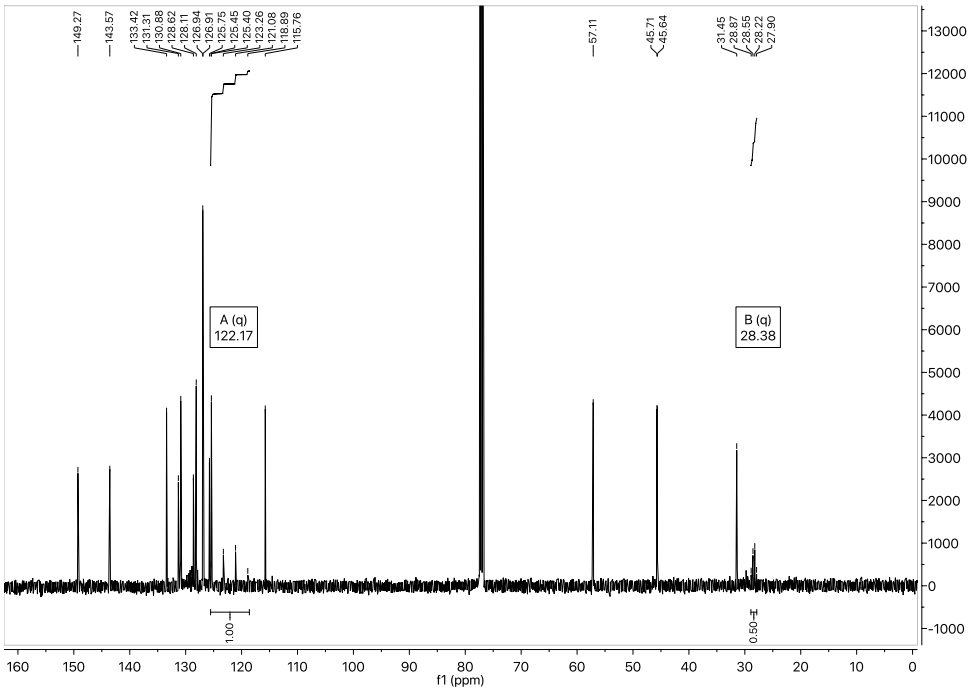


1.2.1.6 4-(4-(3-(Trifluoromethyl)-3H-diazirin-3-yl)phenyl)-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinoline-8-sulfonamide (4TFD-TQS)

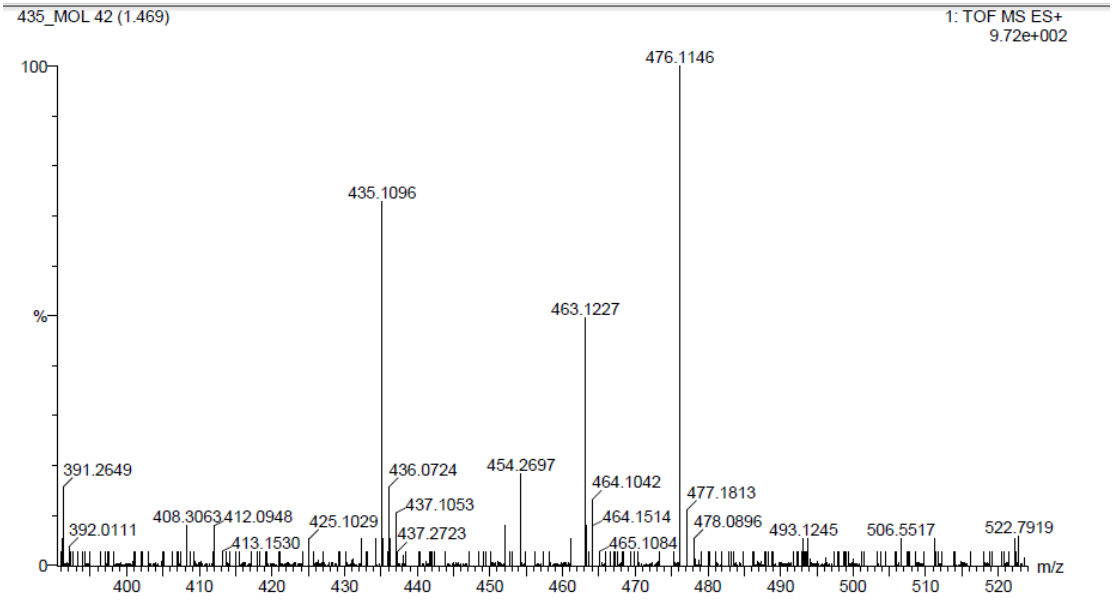
1.2.1.6.1 ^{1}H NMR



1.2.1.6.2 ^{13}C NMR

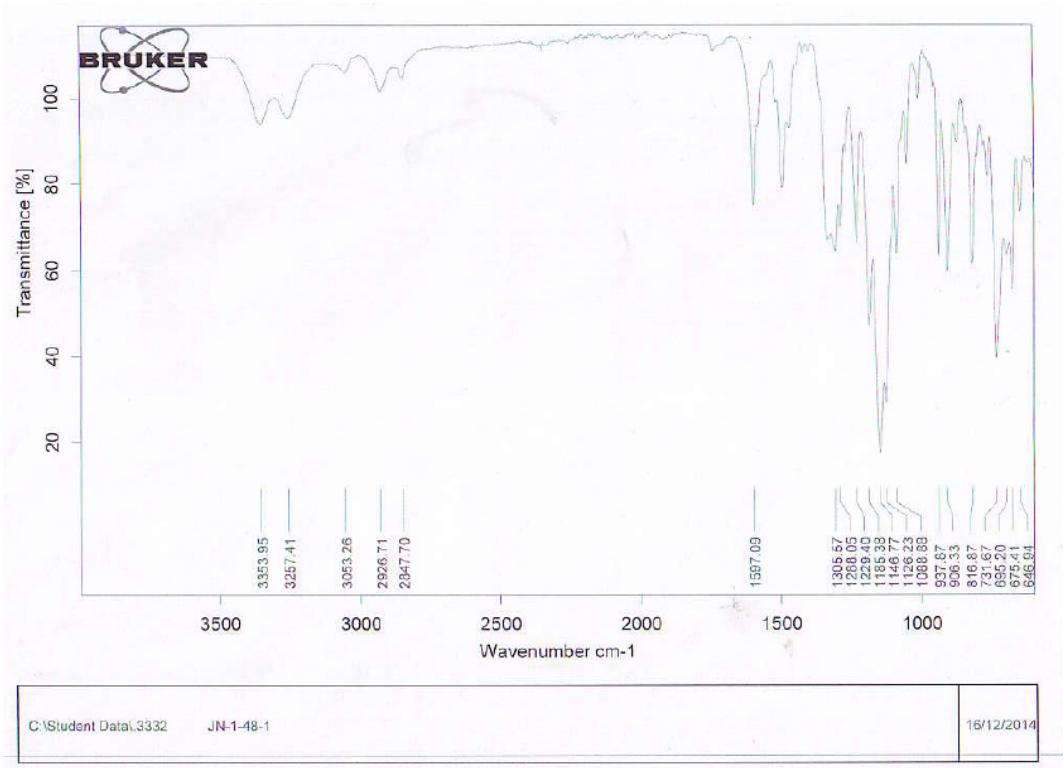


1.2.1.6.3 HRMS



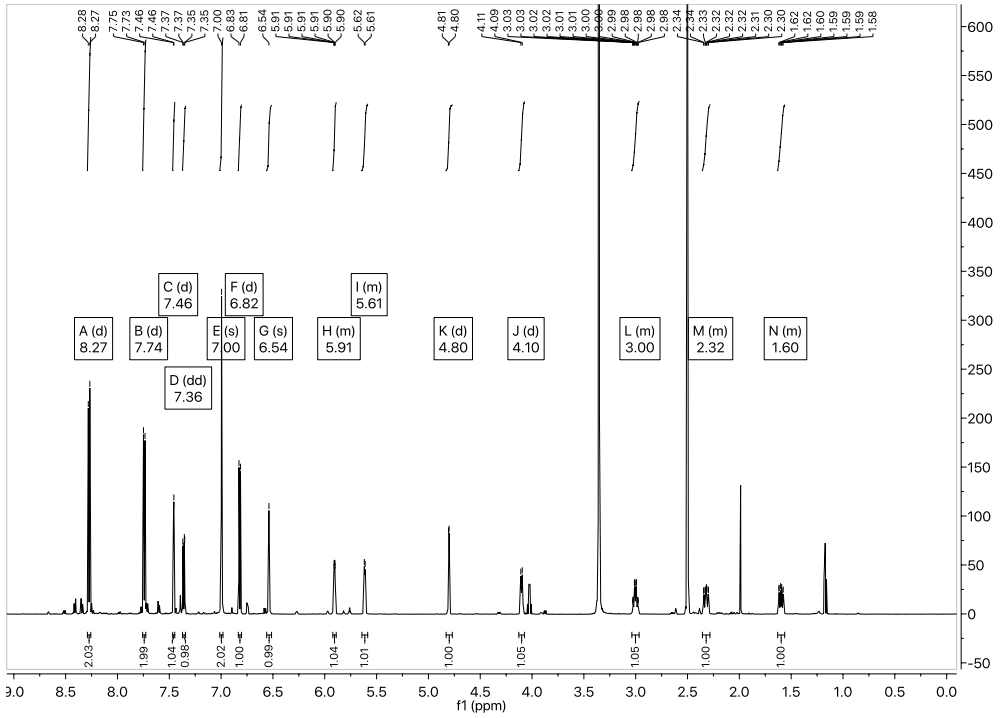
Minimum:				-1.5		
Maximum:		5.0	5.0	50.0		
Mass	Calc. Mass	mDa	PPM	DBE	i-FIT	Formula
435.1096	435.1103	-0.7	-1.6	12.5	n/a	C20 H18 N4 O2 S F3

1.2.1.6.4 IR

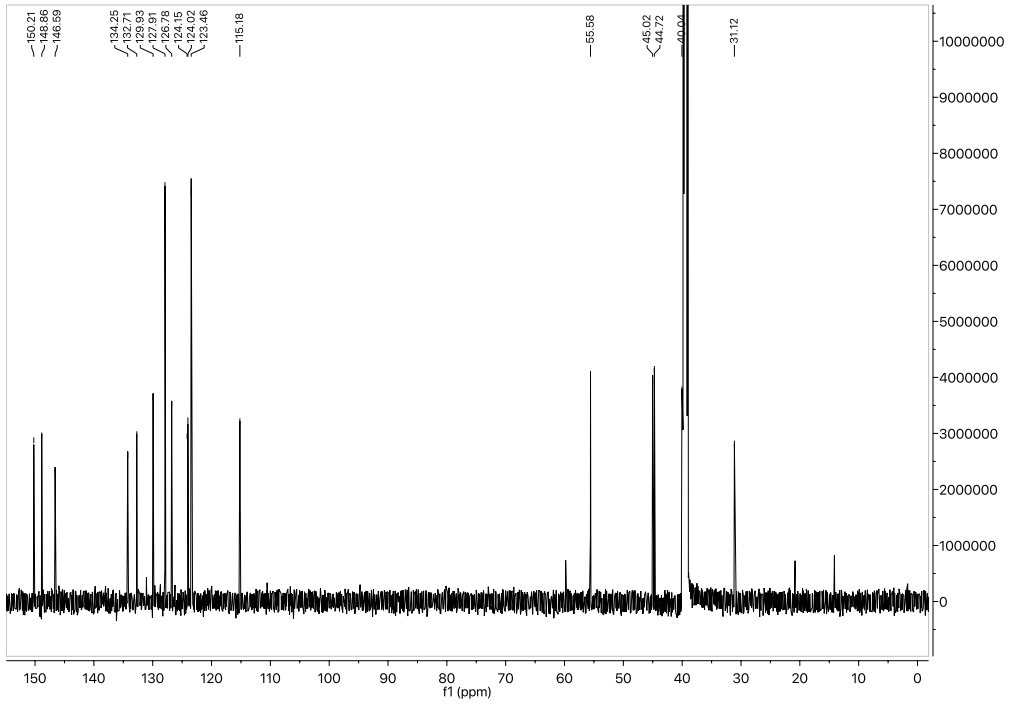


1.2.1.7 4-(4-Nitrophenyl)-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinoline-8-sulfonamide

1.2.1.7.1 ^{1}H NMR



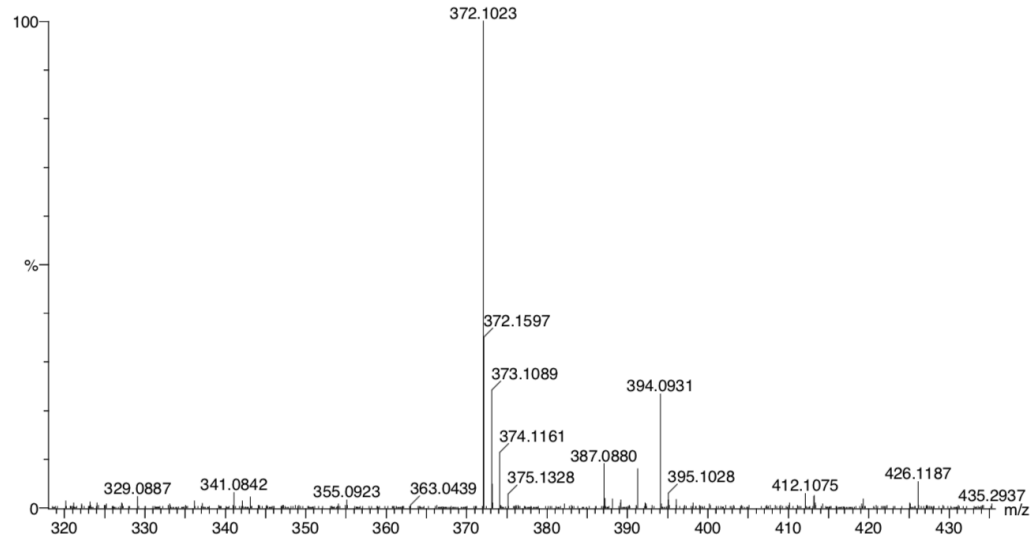
1.2.1.7.2 ^{13}C NMR



1.2.1.7.3 HRMS

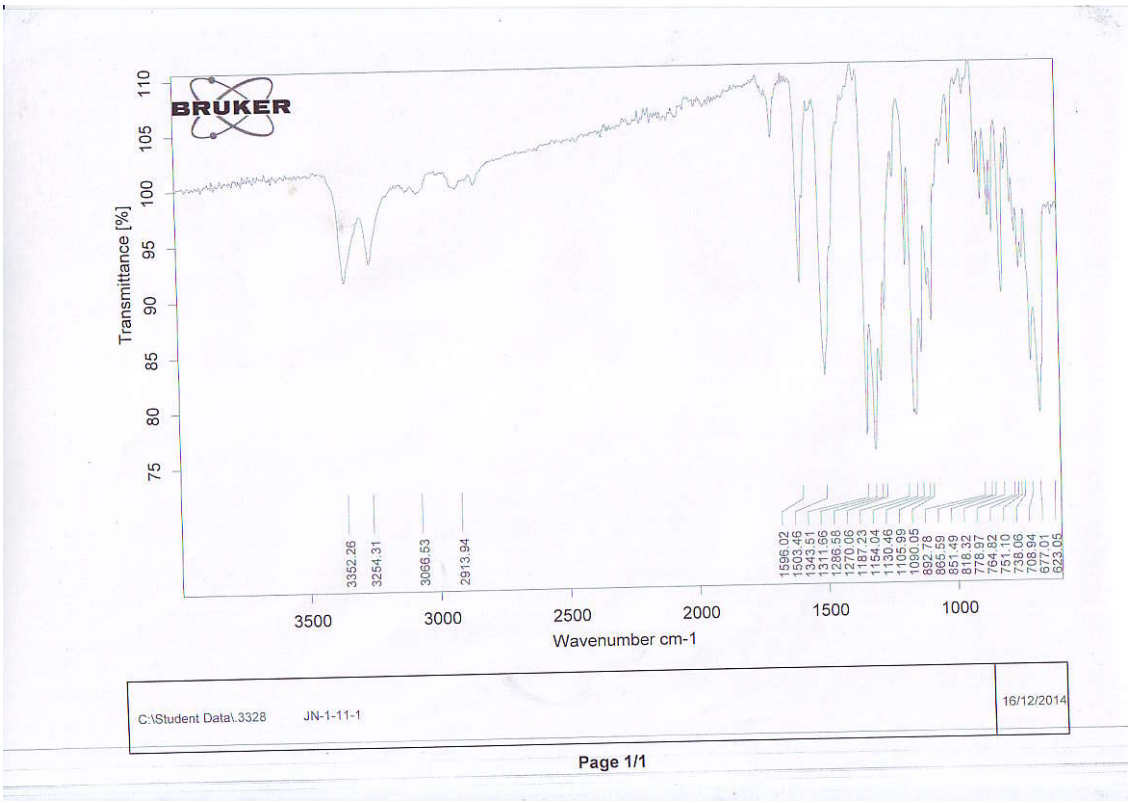
Elements Used:
C: 0-20 H: 0-25 N: 0-3 O: 0-5 S: 0-6
JN1111
JN_1_11_1 34 (1.188)

1: TOF MS ES+
9.60e+003



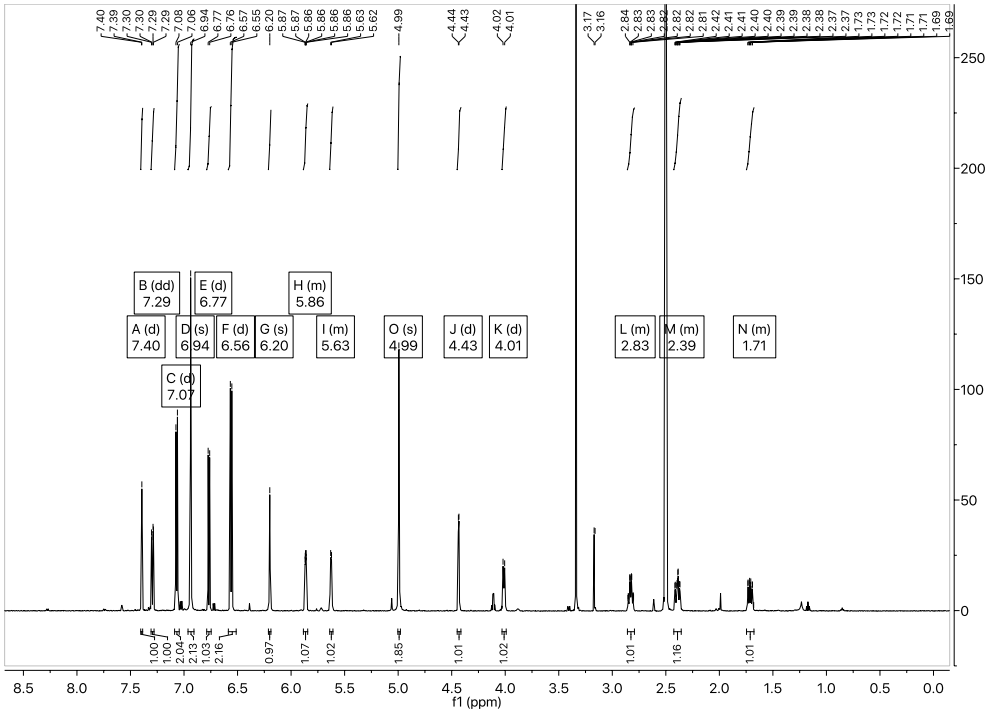
Minimum:				-1.5		
Maximum:		5.0	20.0	50.0		
Mass	Calc. Mass	mDa	PPM	DBE	i-FIT	Formula
372.1023	372.1018	0.5	1.3	11.5	59.9	C18 H18 N3 O4 S
	372.1052	-2.9	-7.8	6.5	37.8	C15 H22 N3 O4 S2
	372.1092	-6.9	-18.5	10.5	14.8	C20 H22 N O2 S2

1.2.1.7.4 IR

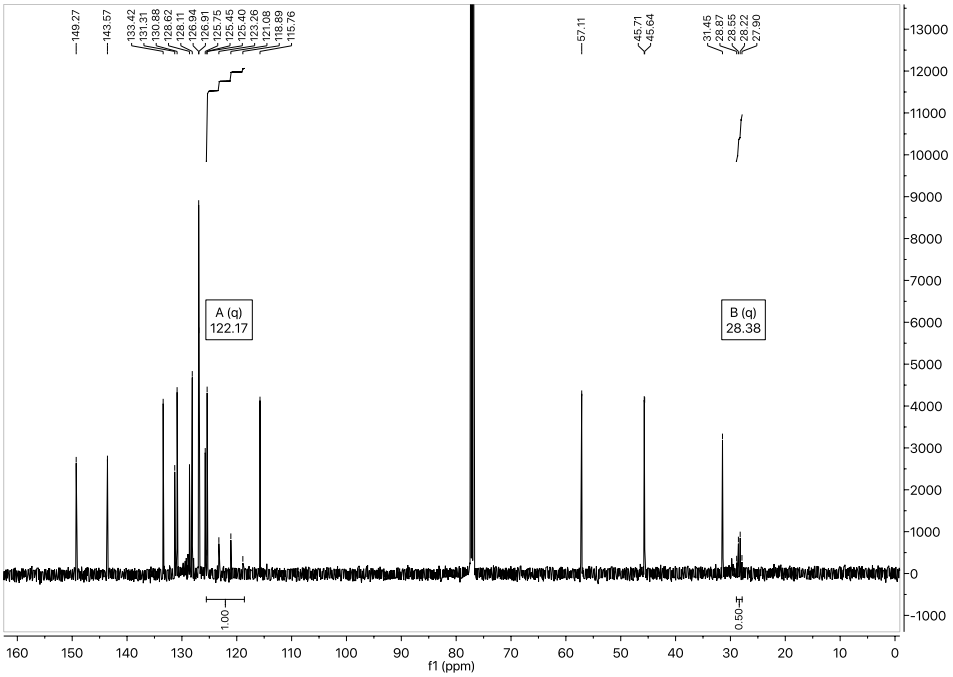


1.2.1.8 4-(4-Aminophenyl)-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinoline-8-sulfonamide

1.2.1.8.1 ^{1}H NMR

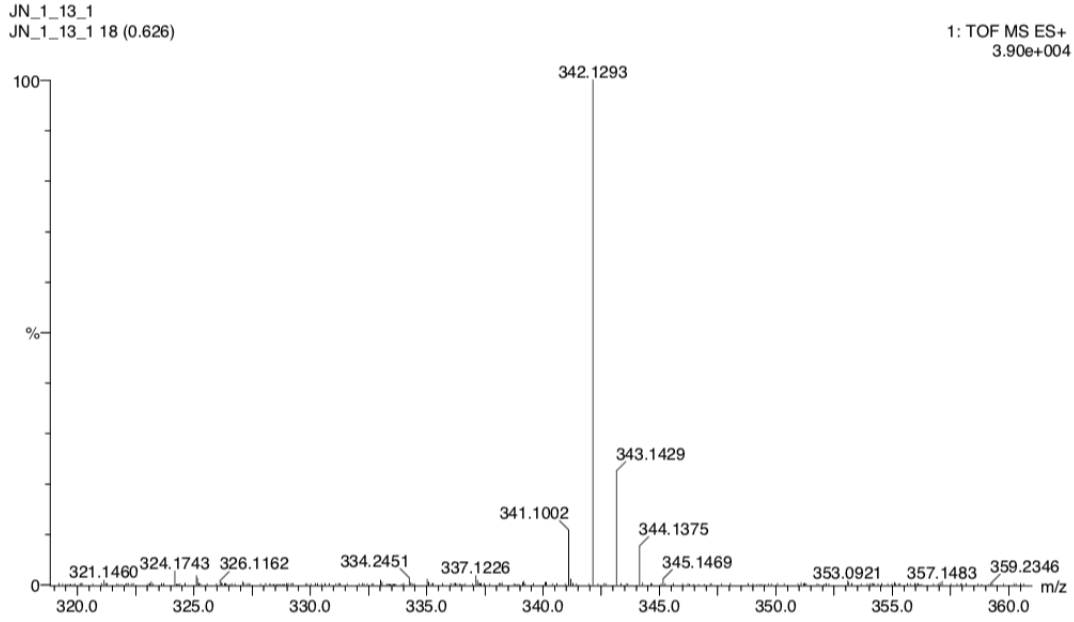


1.2.1.8.2 ^{13}C NMR



1.2.1.8.3 HRMS

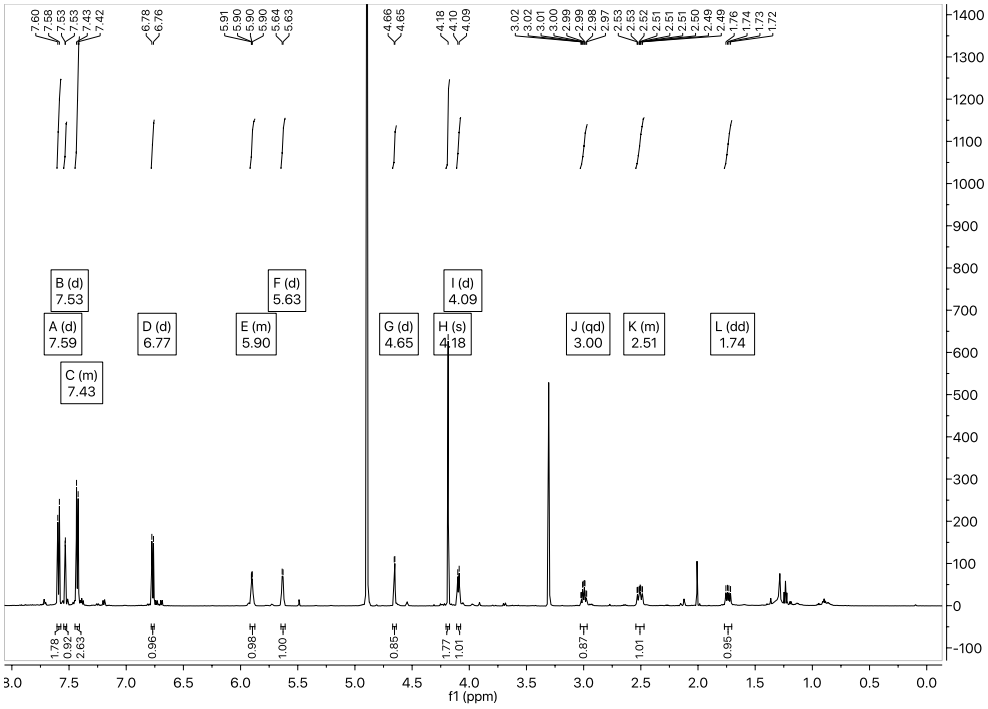
Monoisotopic Mass, Even Electron Ions
197 formula(e) evaluated with 2 results within limits (up to 50 best isotopic matches for each mass)
Elements Used:
C: 0-20 H: 0-25 N: 0-3 O: 0-3 S: 0-6
JN_1_13_1
JN_1_13_1 18 (0.626)



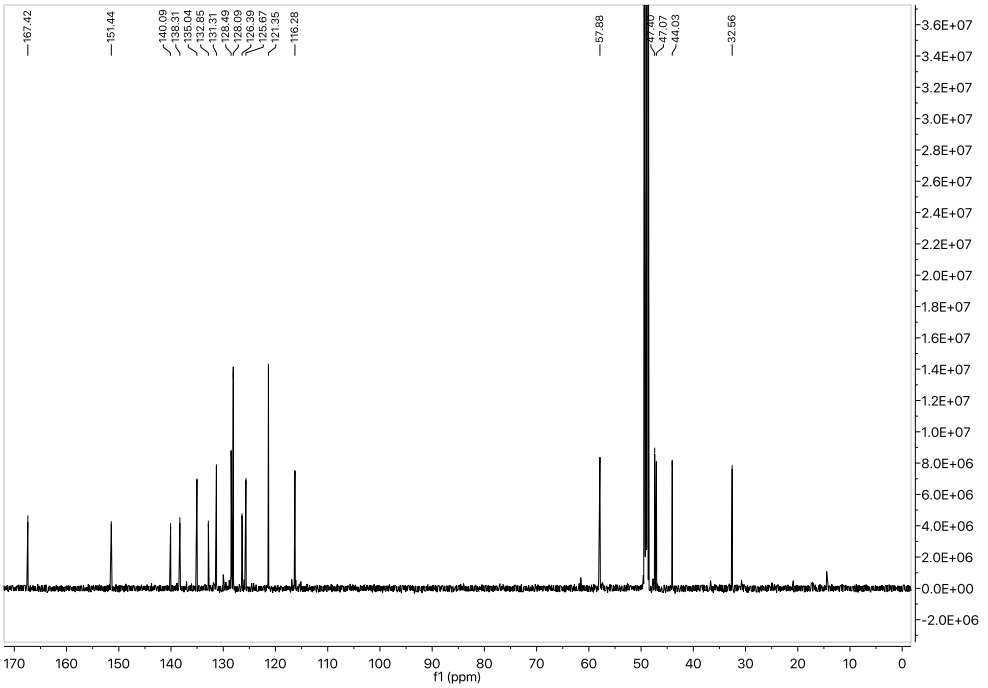
Minimum:				-1.5		
Maximum:		5.0	10.0	50.0		
Mass	Calc. Mass	mDa	PPM	DBE	i-FIT	Formula
342.1293	342.1276	1.7	5.0	10.5	43.5	C18 H20 N3 O2 S
	342.1310	-1.7	-5.0	5.5	410.8	C15 H24 N3 O2 S2

1.2.1.9 2-Chloro-N-(4-(8-sulfamoyl-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinolin-4-yl)phenyl)acetamide

1.2.1.9.1 ^{1}H NMR



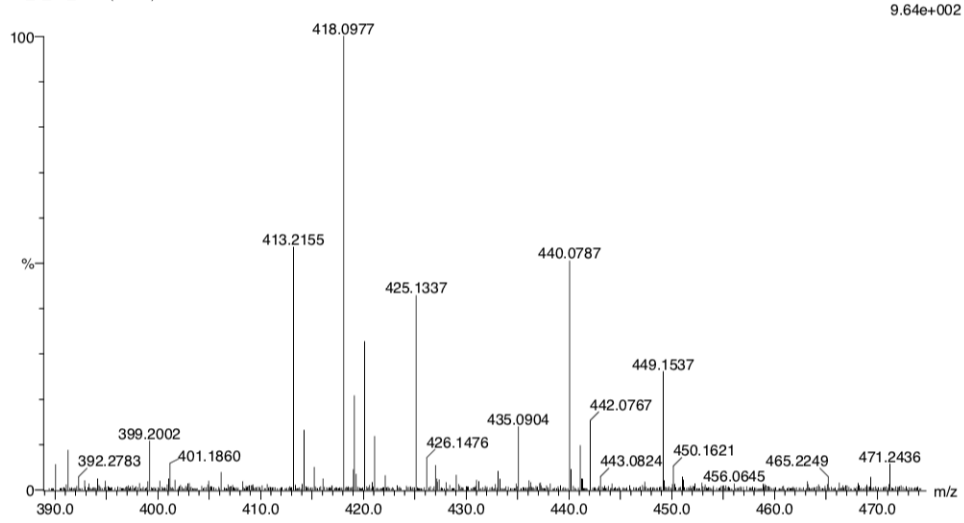
1.2.1.9.2 ^{13}C NMR



1.2.1.9.3 HRMS

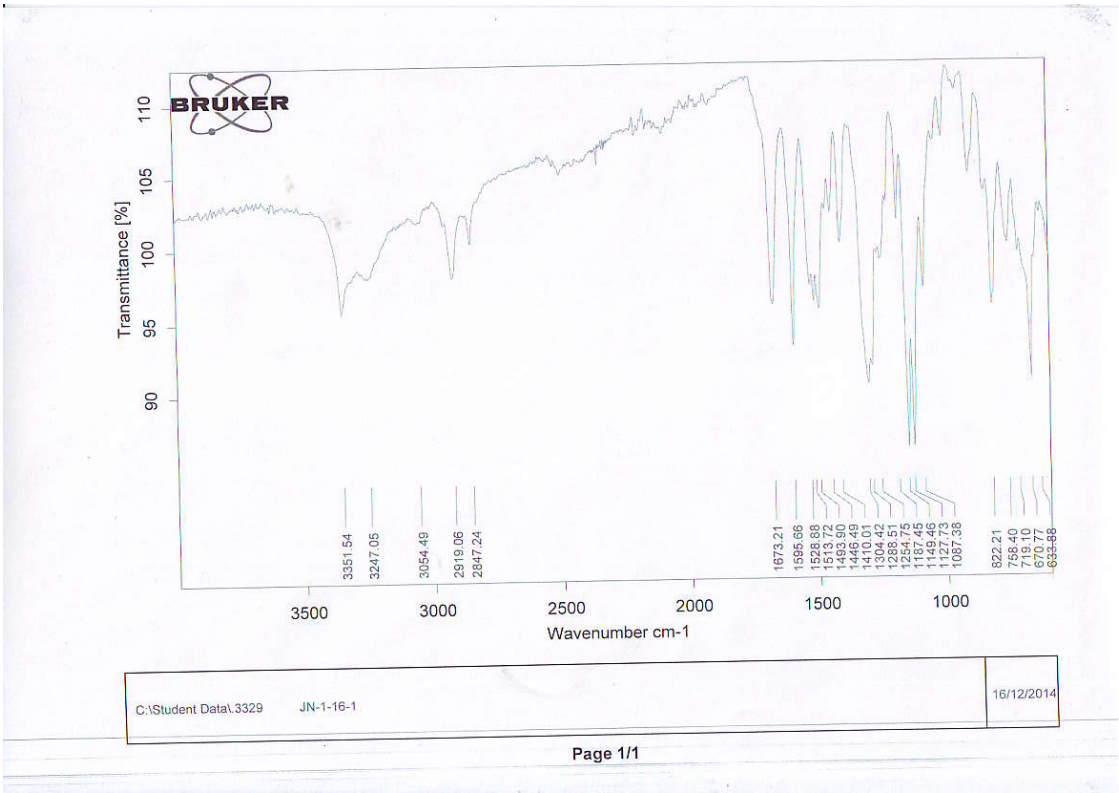
Monoisotopic Mass, Even Electron Ions
105 formula(e) evaluated with 3 results within limits (up to 50 best isotopic matches for each mass)
Elements Used:
C: 20-23 H: 0-25 N: 0-3 O: 0-3 Na: 0-1 S: 0-1 Cl: 0-1
JN_1_16_1
JN_1_16_3 50 (1.747)

1: TOF MS ES+
9.64e+002



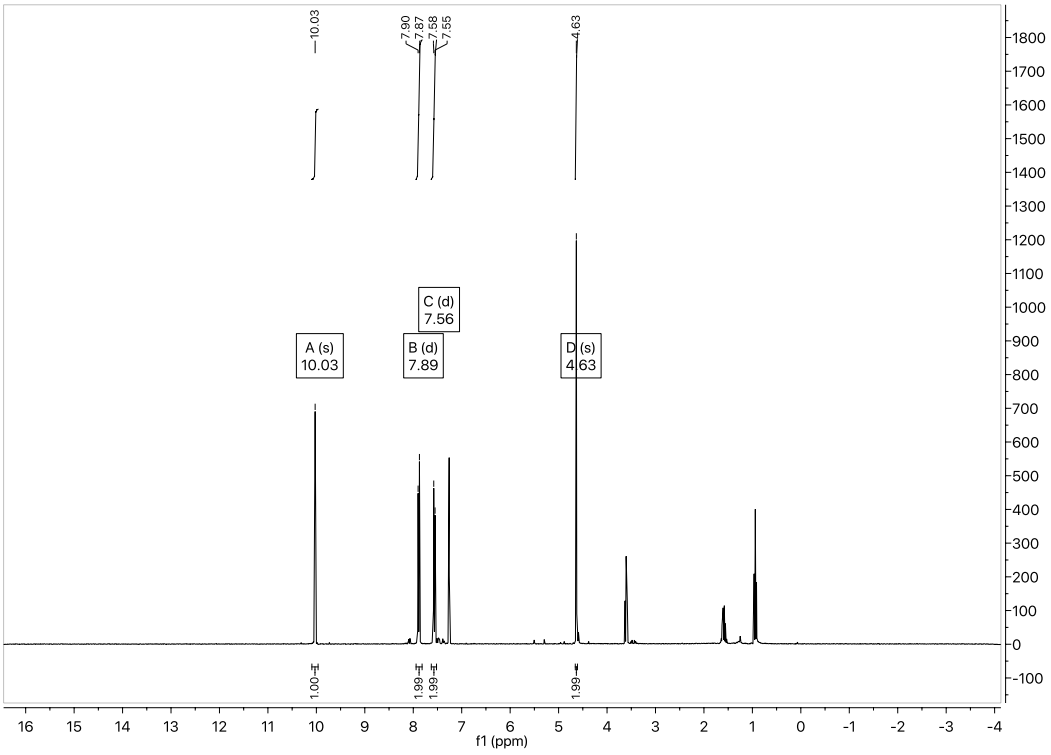
Minimum:		5.0	10.0	-1.5		
Maximum:				50.0		
Mass	Calc. Mass	mDa	PPM	DBE	i-FIT	Formula
418.0977	418.0992	-1.5	-3.6	11.5	6.6	C20 H21 N3 O3 S Cl
	418.0958	1.9	4.5	16.5	7.3	C23 H17 N3 O3 Cl
	418.1008	-3.1	-7.4	12.5	10.3	C23 H22 N O Na S Cl

1.2.1.9.4 IR

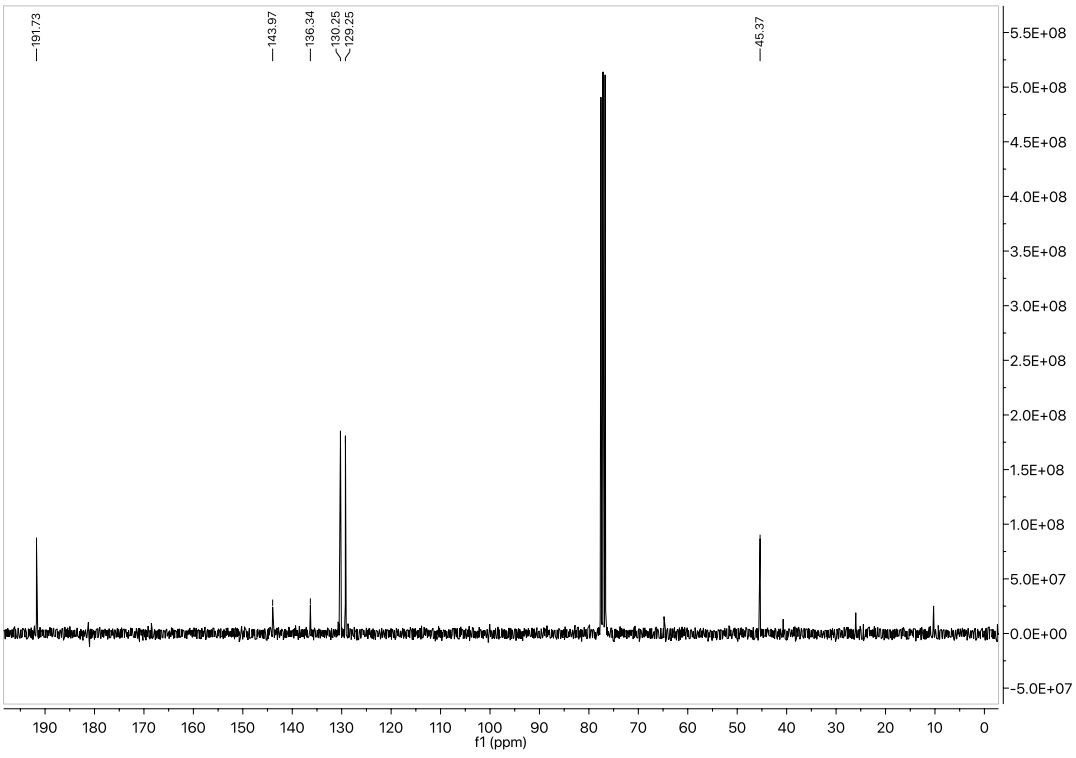


1.2.1.10 4-(Chloromethyl)benzaldehyde

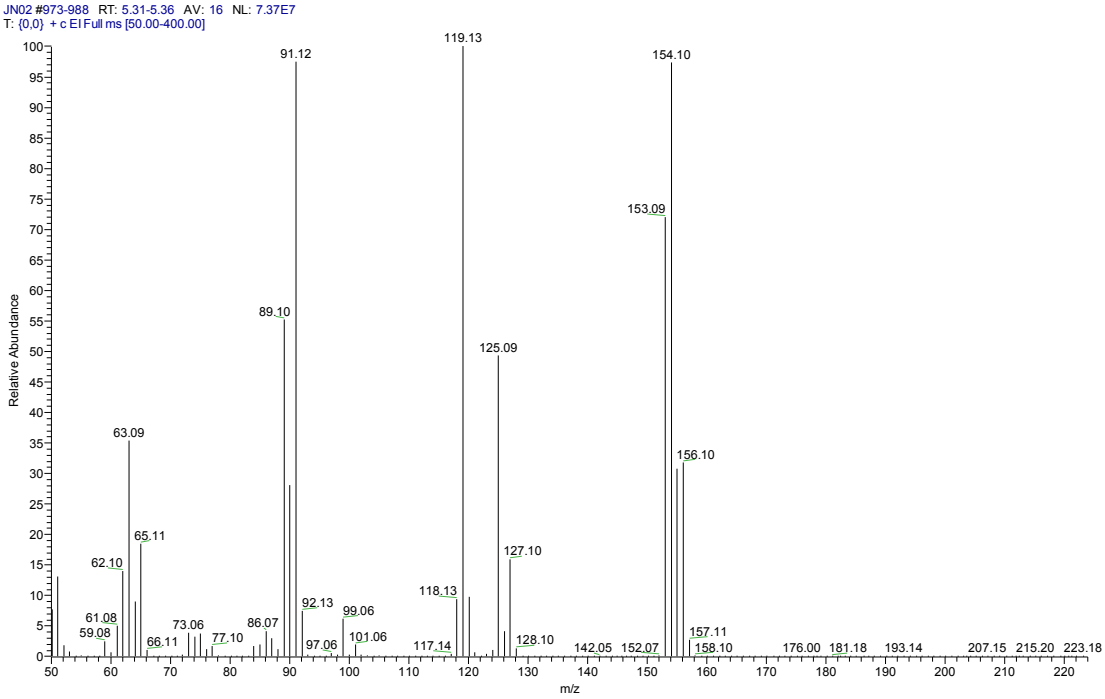
1.2.1.10.1 ^{1}H NMR



1.2.1.10.2 ^{13}C NMR

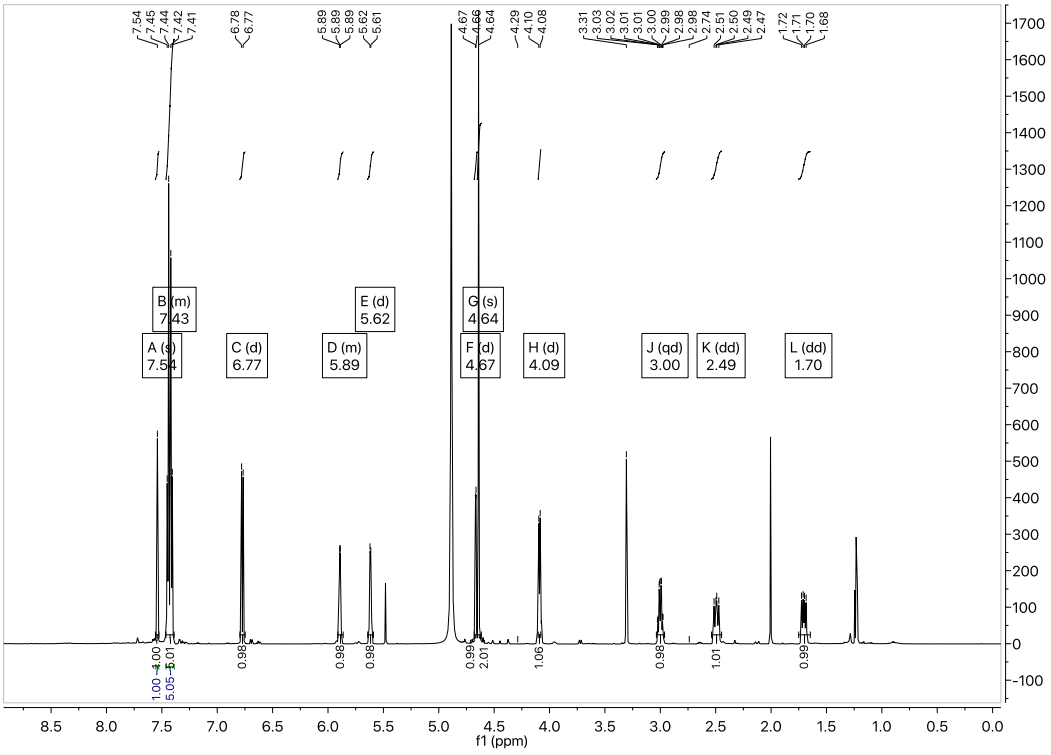


1.2.1.10.3 LRMS

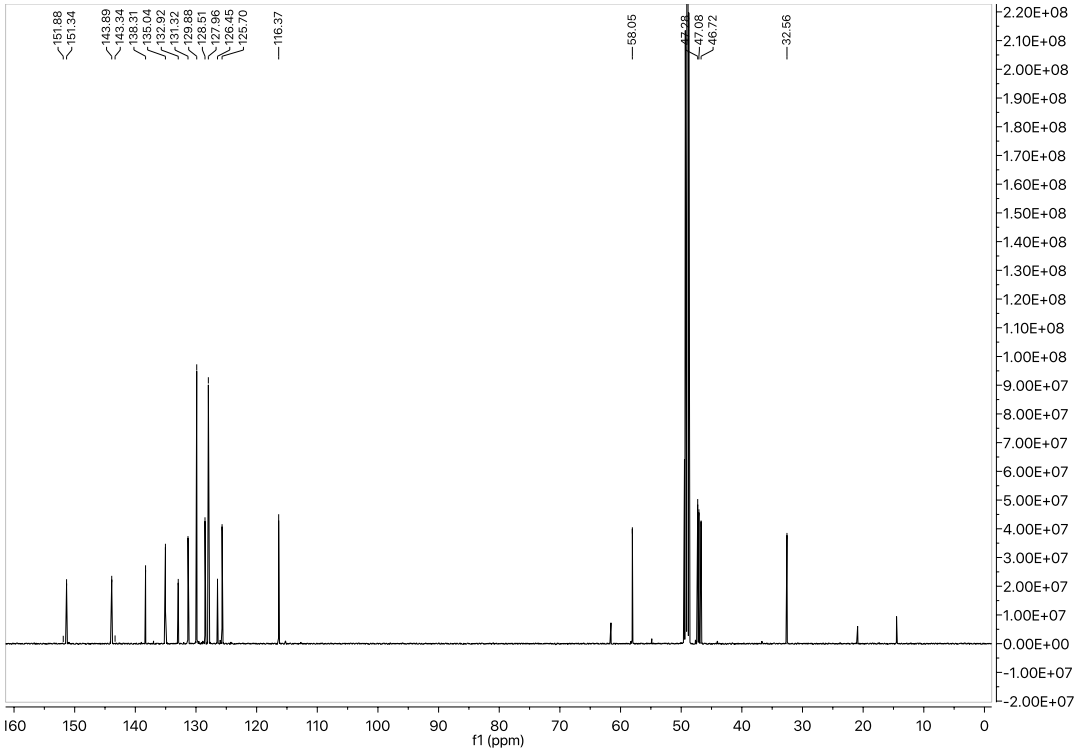


1.2.1.11 4-(4-(Chloromethyl)phenyl)-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinoline-8-sulfonamide

1.2.1.11.1 ^{1}H NMR

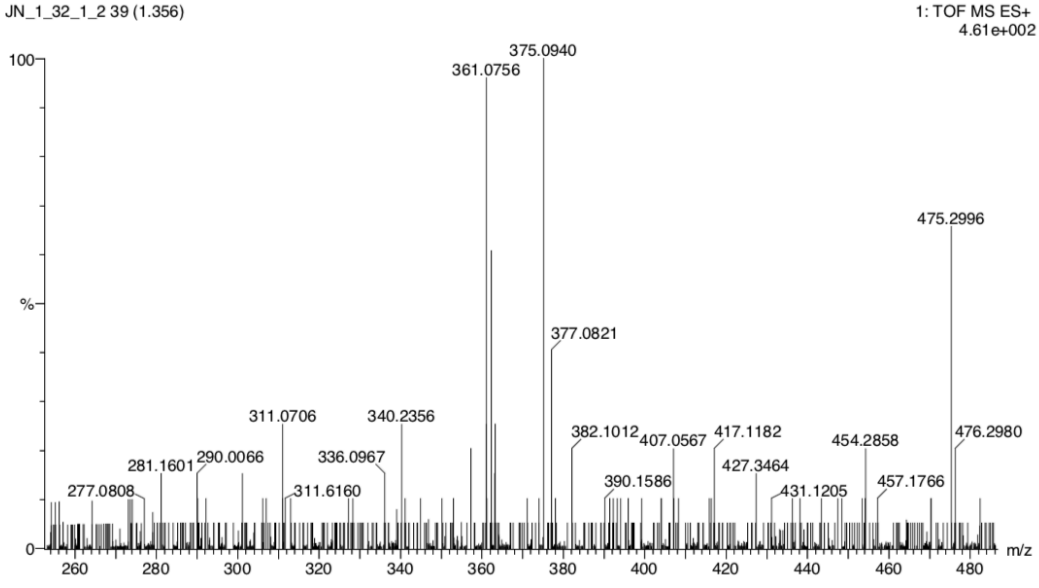


1.2.1.11.2 ^{13}C NMR



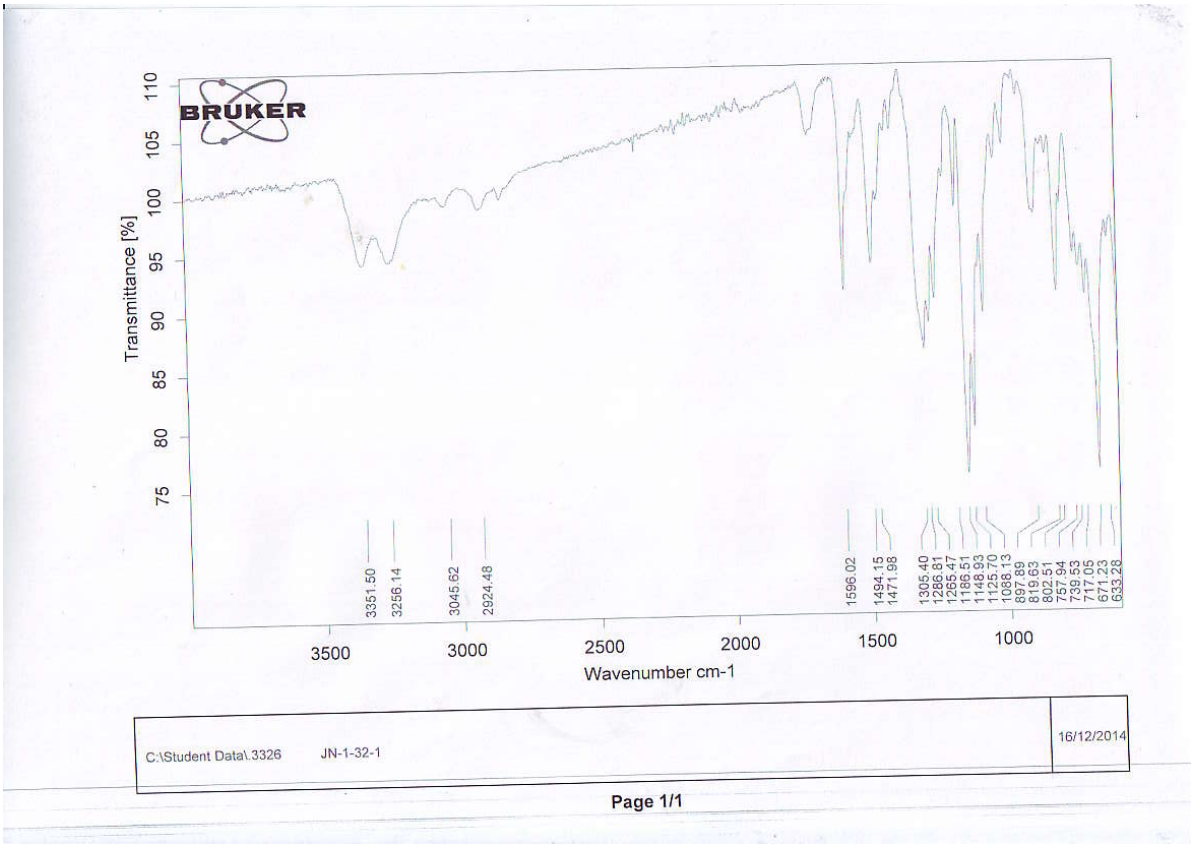
1.2.1.11.3 HRMS

Monoisotopic Mass, Even Electron Ions
124 formula(e) evaluated with 1 results within limits (all results (up to 1000) for each mass)
Elements Used:
C: 0-20 H: 0-20 N: 0-2 O: 0-2 S: 0-2 Cl: 0-3
JN1321
JN_1_32_1_2 39 (1.356)



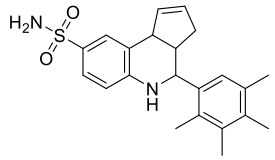
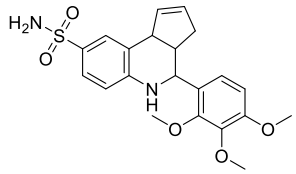
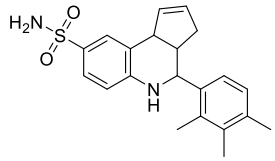
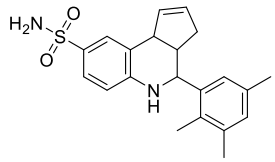
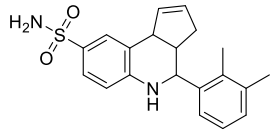
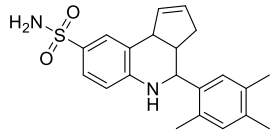
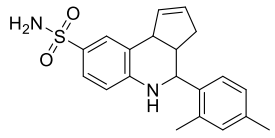
Minimum:		5.0	5.0	-1.5		
Maximum:				50.0		
Mass	Calc. Mass	mDa	PPM	DBE	i-FIT	Formula
375.0940	375.0934	0.6	1.6	10.5	85.8	C19 H20 N2 O2 S Cl

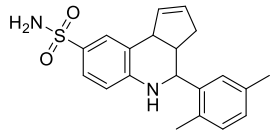
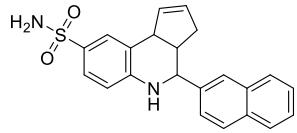
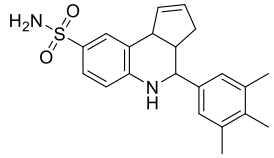
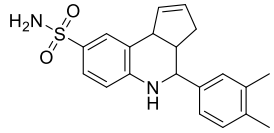
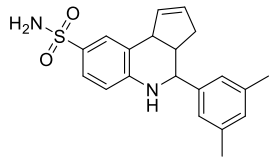
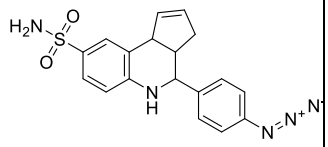
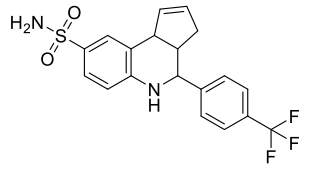
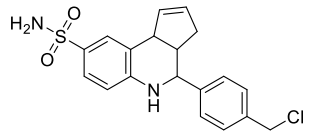
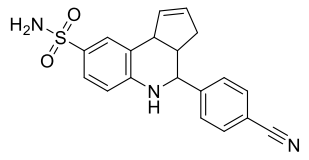
1.2.1.11.4 IR

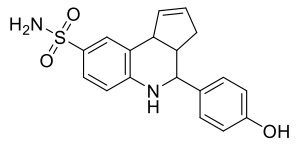
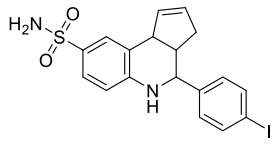
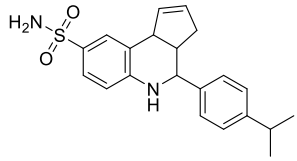
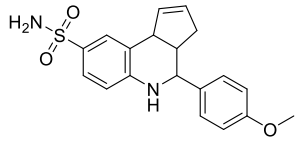
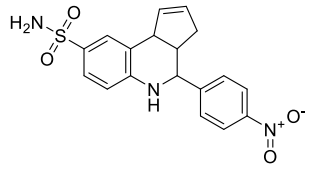
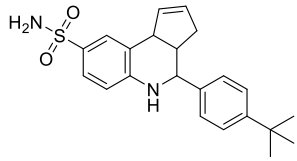
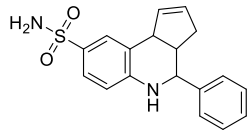
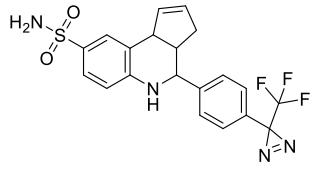

BRUKER
Transmittance [%]
110
105
100
95
90
85
80
75
3351.50
3256.14
3045.62
2924.48
1596.02
1494.15
1471.98
1305.40
1286.81
1265.47
1186.51
1148.93
1125.70
1088.13
897.89
819.63
802.51
757.94
739.53
717.05
671.23
633.28
3500
3000
2500
2000
1500
1000
Wavenumber cm-1
C:\Student Data\.3326
JN-1-32-1
16/12/2014
Page 1/1

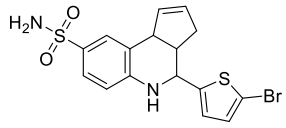
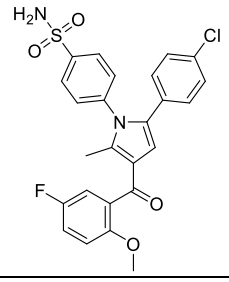
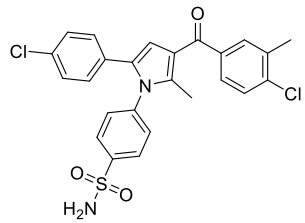
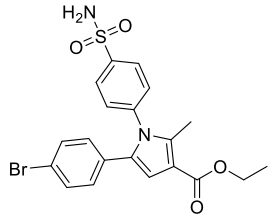
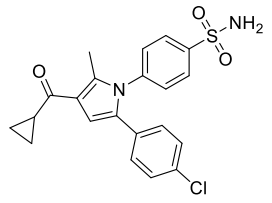
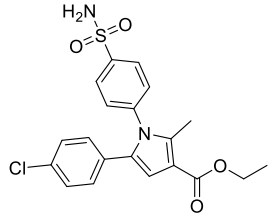
1.3 Appendix – Chapter 5

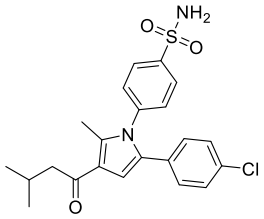
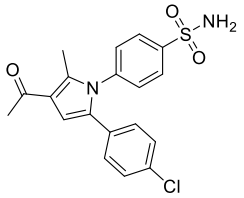
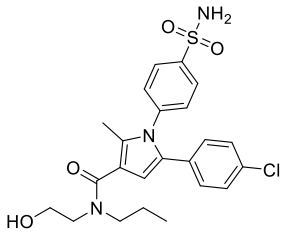
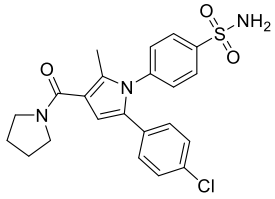
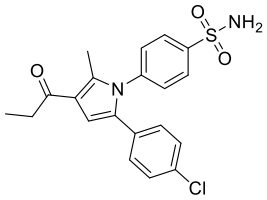
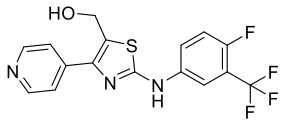
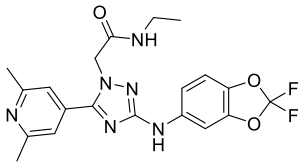
1.3.1 Chapter 5 – Appendix 1: Compounds used in the actives database

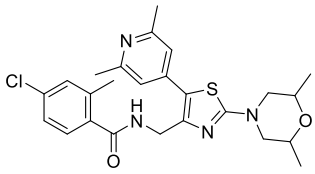
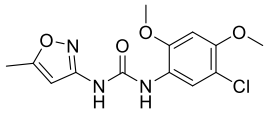
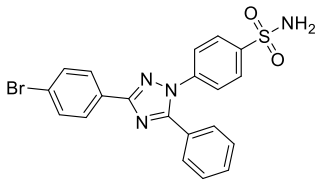
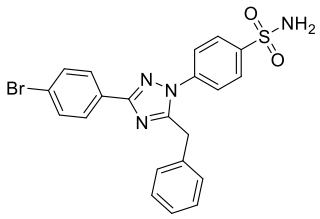
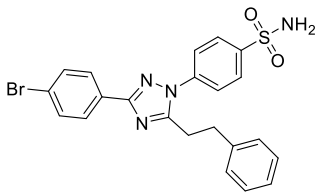
Structure	Name	Reference
TQS-family		
	4-(2,3,4,5-tetramethylphenyl)-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinoline-8-sulfonamide	(1)
	4-(2,3,4-trimethoxyphenyl)-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinoline-8-sulfonamide	*Unpublished*
	4-(2,3,4-trimethylphenyl)-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinoline-8-sulfonamide	(1)
	4-(2,3,5-trimethylphenyl)-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinoline-8-sulfonamide	(1)
	4-(2,3-dimethylphenyl)-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinoline-8-sulfonamide	(1)
	4-(2,4,5-trimethylphenyl)-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinoline-8-sulfonamide	(1)
	4-(2,4-dimethylphenyl)-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinoline-8-sulfonamide	(1)

	4-(2,5-dimethylphenyl)-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinoline-8-sulfonamide	(1)
	4-(naphthalen-2-yl)-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinoline-8-sulfonamide	(2)
	4-(3,4,5-trimethylphenyl)-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinoline-8-sulfonamide	(1)
	4-(3,4-dimethylphenyl)-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinoline-8-sulfonamide	(1)
	4-(3,5-dimethylphenyl)-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinoline-8-sulfonamide	(1)
	4-(4-azidophenyl)-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinoline-8-sulfonamide	*Unpublished*
	4-(4-(trifluoromethyl)phenyl)-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinoline-8-sulfonamide	(2)
	4-(4-(chloromethyl)phenyl)-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinoline-8-sulfonamide	*Unpublished*
	4-(4-cyanophenyl)-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinoline-8-sulfonamide	*Unpublished*

	4-(4-hydroxyphenyl)-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinoline-8-sulfonamide	(2)
	4-(4-iodophenyl)-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinoline-8-sulfonamide	(2)
	4-(4-isopropylphenyl)-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinoline-8-sulfonamide	*Unpublished*
	4-(4-methoxyphenyl)-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinoline-8-sulfonamide	*Unpublished*
	4-(4-nitrophenyl)-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinoline-8-sulfonamide	*Unpublished*
	4-(4-(tert-butyl)phenyl)-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinoline-8-sulfonamide	*Unpublished*
	4-phenyl-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinoline-8-sulfonamide	(2)
	4-(4-(3-(trifluoromethyl)-3H-diazirin-3-yl)phenyl)-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinoline-8-sulfonamide	*Unpublished*

	4-(5-bromothiophen-2-yl)-3a,4,5,9b-tetrahydro-3H-cyclopenta[c]quinoline-8-sulfonamide	(4)
Abbott-family		
	4-(5-(4-chlorophenyl)-3-(5-fluoro-2-methoxybenzoyl)-2-methyl-1H-pyrrol-1-yl)benzenesulfonamide	(3)
	4-(3-(4-chloro-3-methylbenzoyl)-5-(4-chlorophenyl)-2-methyl-1H-pyrrol-1-yl)benzenesulfonamide	(3)
	ethyl 5-(4-bromophenyl)-2-methyl-1-(4-sulfamoylphenyl)-1H-pyrrole-3-carboxylate	(3)
	4-(5-(4-chlorophenyl)-3-(cyclopropanecarbonyl)-2-methyl-1H-pyrrol-1-yl)benzenesulfonamide	(3)
	ethyl 5-(4-chlorophenyl)-2-methyl-1-(4-sulfamoylphenyl)-1H-pyrrole-3-carboxylate	(3)

	4-(5-(4-chlorophenyl)-2-methyl-3-(3-methylbutanoyl)-1H-pyrrol-1-yl)benzenesulfonamide	(3)
	4-(3-acetyl-5-(4-chlorophenyl)-2-methyl-1H-pyrrol-1-yl)benzenesulfonamide	(3)
	5-(4-chlorophenyl)-N-(2-hydroxyethyl)-2-methyl-N-propyl-1-(4-sulfamoylphenyl)-1H-pyrrole-3-carboxamide	(3)
	4-(5-(4-chlorophenyl)-2-methyl-3-(pyrrolidine-1-carbonyl)-1H-pyrrol-1-yl)benzenesulfonamide	(3)
	4-(5-(4-chlorophenyl)-2-methyl-3-propionyl-1H-pyrrol-1-yl)benzenesulfonamide	(3)
Janssen-family		
	(2-((4-fluoro-3-(trifluoromethyl)phenyl)amino)-4-(pyridin-4-yl)thiazol-5-yl)methanol	(5)
	2-(3-((2,2-difluorobenzo[d][1,3]dioxol-5-yl)amino)-5-(2,6-dimethylpyridin-4-yl)-1H-1,2,4-triazol-1-yl)-N-ethylacetamide	(6)

	4-chloro-N-((2-(2,6-dimethylmorpholino)-5-(2,6-dimethylpyridin-4-yl)thiazol-4-yl)methyl)-2-methylbenzamide	(7)
Pfizer-family		
	1-(5-chloro-2,4-dimethoxyphenyl)-3-(5-methylisoxazol-3-yl)urea	(8)
TBS-family		
	4-(3-(4-bromophenyl)-5-phenyl-1H-1,2,4-triazol-1-yl)benzenesulfonamide	(9)
	4-(5-benzyl-3-(4-bromophenyl)-1H-1,2,4-triazol-1-yl)benzenesulfonamide	(9)
	4-(3-(4-bromophenyl)-5-phenethyl-1H-1,2,4-triazol-1-yl)benzenesulfonamide	(9)

1. Gill-Thind JK, Dhankher P, D'Oyley JM, Sheppard TD, Millar NS (2015) Structurally Similar Allosteric Modulators of α7 Nicotinic Acetylcholine Receptors Exhibit Five Distinct Pharmacological Effects. *J Biol Chem* 290(6):3552–3562.

2. Gill JK, Dhankher P, Sheppard TD, Sher E, Millar NS (2012) A series of α7 nicotinic acetylcholine receptor allosteric modulators with close chemical similarity but diverse pharmacological properties. *Mol Pharmacol* 81(5):710–8.

3. Thakur GA, Kulkarni AR, Deschamps JR, Papke RL (2013) Expeditious synthesis, enantiomeric resolution, and enantiomer functional characterization of (4-(4-bromophenyl)-3a,4,5,9b-tetrahydro-3 h -cyclopenta[c]quinoline-8-sulfonamide (4BP-TQS): An allosteric agonist-positive allosteric modulator of alpha7 nAChR. *J Med Chem* 56(21):8943–8947.

4. Faghih R, et al. (2009) Discovery of 4-(5-(4-Chlorophenyl)-2-methyl-3-propionyl-1 H -pyrrol-1-yl)benzenesulfonamide (A-867744) as a Novel Positive Allosteric Modulator of the α7 Nicotinic Acetylcholine Receptor. *J Med Chem* 52(10):3377–3384.

5. Dinklo T, et al. (2011) 4- (4-pyridinyl) -5-thiazolemethanol (JNJ-1930942), a Novel Positive Allosteric Modulator of the α7 Nicotinic Acetylcholine Receptor □. *Pharmacology* 1:560–574.

6. Janssen Pharmaceutica NV; Thuring J, et al. (2007) 2-Aniline-4-aryl substituted thiazole derivatives; WO 2009/115547 A1,

7. Janssen Pharmaceutica NV; Macdonald G. J., De Boeck B. C. A. G., Leenaerts J. E. (2011) Morpholinothiazoles as alpha 7 positive allosteric modulators; WO 2011/064288 A1

8. Hurst RS, et al. (2005) A novel positive allosteric modulator of the alpha7 neuronal nicotinic acetylcholine receptor: in vitro and in vivo characterization. *J Neurosci* 25(17):4396–4405.

9. Chatzidaki A, D'Oyley JM, Gill-Thind JK, Sheppard TD, Millar NS (2015) The influence of allosteric modulators and transmembrane mutations on desensitisation and activation of α7 nicotinic acetylcholine receptors. *Neuropharmacology* 97:75–85.

1.3.2 Chapter 5 –Appendix 2: Python script for determination of the CNS MPO scores for compounds listed in a table with their corresponding physicochemical properties

```
#!usr/bin/python
#python implementation of CNS MPO approach to selecting molecules
for synthesis

#get's database IDs, tanimoto combo scores and calculates
paramters for CNS MPO
#then uses these to rank the molecules on drug 'likeness' out of
6
#and similarity to the pharmacophore (ROCS) query (tanimoto
combo) out of 2

import os


#-----------------------------------------------------------------
-

def score_single_decay(prop, des, undes):
      '''
      generate desirability scores for parameters that are
      represented with a monotonic decreasing function
```

```
      '''
      if prop == 'NULL':
            score = 0
      elif prop < des:
            score = 1
      elif prop > undes:
            score = 0
      elif prop >= des and prop <= undes:
            m = (-1)/(undes - des)
            c = 1 - (m * des)
            #print m, c
            score = m * prop + c
      #print prop,"score", score
      return score

#-----------------------------------------------------------------
-

def score_hump(prop, undes_lower, des_lower, des_upper,
undes_upper):
      '''
      generate desirability scores for parameters represented by
      a hump function
      '''
      if prop == 'NULL':
            score = 0
      elif prop < undes_lower or prop > undes_upper:
            score = 0
      elif prop > des_lower and prop < des_upper:
            score = 1
      elif prop >= undes_lower and prop <= des_lower:
            m = (1)/(des_lower - undes_lower)
            c = 1 - (m * des_lower)
            score = m * prop + c
      elif prop >= des_upper and prop <= undes_upper:
            m = (-1)/(undes_upper - des_upper)
            c = 1 - (m * des_upper)
            score = m * prop + c
      #print prop,'score',score
      return score

#-----------------------------------------------------------------
-

def grab_value(item):
      '''
      get items from combined_data.txt file and set as float for
manipulation
      by scoring functions.  If it is not a valid number, sets
value to 'NULL'
      which results in a score of 0 from the MPO scoring functions
      '''
      try:
            var = float(item)
      except ValueError:
            var = 'NULL'
            next
      return var
```

```
#-----------------------------------------------------------------
-

'''
Region bounds for monotonic functions for each of the 6 CNS MPO
parameters.
As defined in ACS Chem. neurosci. 2010, 1, 6, 435
'''

clogP_des = 3.0
clogP_undes = 5.0
clogD_des = 2.00
clogD_undes = 4.0
MW_des = 360.0
MW_undes = 500.0
TPSA_undes_lower = 20.0
TPSA_des_lower = 40.0
TPSA_des_upper = 90.0
TPSA_undes_upper = 120.0
HBD_des = 0.0
HBD_undes = 4.0
basicpKa_des = 8.0
basicpKa_undes = 10.0

#-----------------------------------------------------------------
-

files = os.listdir('.')

if not 'hit_IDs.txt' in files:
      os.system("python get_ids.py > hit_IDs.txt")
if not 'CNS_MPO_Params.txt' in files:
      os.system("evaluate -e
\"logp();logd('7.4');mass();psa('7.4');donorcount('7.4');pka('1')
\" Drugbank_ROCS_T2_hits_1.sdf > CNS_MPO_Params.txt" )
if not 'combined_data.txt' in files:
      with open('CNS_MPO_Params.txt','r') as pars,
open('hit_IDs.txt','r') as names, open('combined_data.txt','w')
as f_out:
            for t in zip(names,pars):
                  f_out.write(';'.join(x.strip() for x in t) +
'\n')

with open('combined_data.txt','r') as in_file:
      for i, line in enumerate(in_file):
            ln = line.split(';')

            # get the parameter values for each compound
            identifier = ln[0]
            tanimoto_combo = ln[1]
            clogP = grab_value(ln[2])
            clogD = grab_value(ln[3])
            MW = grab_value(ln[4])
            TPSA = grab_value(ln[5])
            HBD = grab_value(ln[6])
            basicpKa = grab_value(ln[7])
            #print identifier,clogP,clogD,MW,TPSA,HBD,basicpKa
```

```
            #calculate the CNS MPO score
            T0_clogP = score_single_decay(clogP, clogP_des,
clogP_undes)
            T0_clogD = score_single_decay(clogD, clogD_des,
clogD_undes)
            T0_MW = score_single_decay(MW, MW_des, MW_undes)
            T0_TPSA = score_hump(TPSA, TPSA_undes_lower,
TPSA_des_lower, TPSA_des_upper, TPSA_undes_upper)
            T0_HBD = score_single_decay(HBD ,HBD_des, HBD_undes)
            T0_basicpKa = score_single_decay(basicpKa,
basicpKa_des, basicpKa_undes)
            CNS_MPO_score = T0_clogP + T0_clogD + T0_MW + T0_TPSA
+ T0_HBD + T0_basicpKa
            if CNS_MPO_score > 4.0 and float(tanimoto_combo) >
0.599:
                  print i+1, identifier, CNS_MPO_score,
tanimoto_combo
```

1.3.3 Chapter 5 – Appendix 3: Table of compounds identified by virtual screening

Rank	DrugBank ID	Compound Name	Target*
1	DB04763	1-N-(4-sulfamoylphenyl-ethyl)-2,4,6-trimethylpyridinium	CAII
2	DB07476	N-[4-(aminosulfonyl)phenyl]-2-mercaptobenzamide	CAII
3	DB08122	N-methyl-4-{[(2-oxo-1,2-dihydro-3h-indol-3-ylidene)methyl]amino}benzenesulfonamide	CDK2
4	DB08202	4-({[(4-methylpiperazin-1-yl)amino]carbonothioyl}amino)benzenesulfonamide	CAII
5	DB04371	AL6528	CAII
6	DB00695	Furosemide	NKCC
7	DB02602	AL7182	CAII
8	DB07257	4-(2-chlorophenyl)-8-(2-hydroxyethyl)-6-methylpyrrolo[3,4-e]indole-1,3(2H,6H)-dione	CAII
9	DB07048	N-[(2R)-5-(aminosulfonyl)-2,3-dihydro-1h-inden-2-yl]-2-propylpentanamide	CAII
10	DB02610	N-(2,3,4,5,6-pentaflouro-benzyl)-4-sulfamoyl-benzamide	CAII
11	DB03294	1-methyl-3-oxo-1,3-dihydro-benzo[c]isothiazole-5-sulfonic acid amide	CAII
12	DB01964	AL5424	CAII
13	DB02221	4-(aminosulfonyl)-N-[(2,4,6-trifluorophenyl)methyl]-benzamide	CAII
14	DB04549	4-(aminosulfonyl)-N-[(2,3,4-trifluorophenyl)methyl]-benzamide	CAII
15	DB03039	4-(aminosulfonyl)-N-[(2,5-difluorophenyl)methyl]-benzamide	CAII
16	DB04180	4-(aminosulfonyl)-N-[(2,4-difluorophenyl)methyl]-benzamide	CAII
17	DB03221	AL7099A	CAII
18	DB02220	AL7089A	CAII

19	DB07742	N-(2,3-difluoro-benzyl)-4-sulfamoyl-benzamide	CAII
20	DB03844	N-(2,6-diflouro-benzyl)-4-sulfamoyl-benzamide	CAII
21	DB02069	N-(2-flouro-benzyl)-4-sulfamoyl-benzamide	CAII
22	**DB00487**	**Pefloxacin**	**DNAG**
23	DB03333	(4-sulfamoyl-phenyl)-thiocarbamic acid O-(2-thiophen-3-yl-ethyl) ester	CAII
24	DB03950	(S)-N-(3-indol-1-yl-2-methyl-propyl)-4-sulfamoyl-benzamide	CAII
25	DB02479	(R)-N-(3-indol-1-yl-2-methyl-propyl)-4-sulfamoyl-benzamide	CAII
26	DB07791	4-{[4-(1-cyclopropyl-2-methyl-1h-imidazol-5-yl)pyrimidin-2-yl]amino}-N-methylbenzenesulfonamide	CDK2
27	DB08673	4-[(5-isopropyl-1,3-thiazol-2-yl)amino]benzenesulfonamide	CDK2
28	DB08083	2-(1,3-thiazol-4-yl)-1h-benzimidazole-5-sulfonamide	CAII
29	DB08165	indane-5-sulfonamide	CAII
30	DB07798	(3R)-3-(fluoromethyl)-N-(3,3,3-trifluoropropyl)-1,2,3,4-tetrahydroisoquinoline-7-sulfonamide	PNMT
31	DB06771	Besifloxacin	DNAG
32	DB07115	N-(4-chlorobenzyl)-N-methylbenzene-1,4-disulfonamide	CAXIII
33	DB07363	Thiophene-2,5-disulfonic acid 2-amide-5-(4-methyl-benzylamide)	CAII
34	DB08134	4-[(6-chloropyrazin-2-yl)amino]benzenesulfonamide	CDK2
35	DB01208	Sparfloxacin	DNAG
36	DB01689	Inhibitor Idd 384	AR
37	DB04608	9-hydroxy-4-phenyl-6h-pyrrolo[3,4-c]carbazole-1,3-dione	W1LPK
38	DB02292	Irosustat	CAII
39	DB02197	4-[(4-imidazo[1,2-a]pyridin-3-ylpyrimidin-2-yl)amino]benzenesulfonamide	CDK2
40	DB01059	Norfloxacin	DNAG
41	DB00817	Rosoxacin	DNAG
42	DB08301	N-({[4-(aminosulfonyl)phenyl]amino}carbonyl)-4-methylbenzenesulfonamide	CAII
43	DB02741	CD564	RAR
44	DB05488	Technetium Tc-99m ciprofloxacin	DNAG
45	DB00537	Ciprofloxacin	DNAG
46	DB01657	2-amino-3-[4-hydroxy-6-oxo-3-(2-phenyl-cyclopropylimino)-cyclohexa-1,4-dienyl]-propionic acid	PAO
47	DB03034	D-Levofloxacin	DNAG
48	DB04089	AL5300	CAII
49	DB07226	N-[4-(2-chlorophenyl)-1,3-dioxo-1,2,3,6-tetrahydropyrrolo[3,4-c]carbazol-9-yl]formamide	W1LPK
50	DB08106	N-[(6-butoxynaphthalen-2-yl)sulfonyl]-D-glutamic acid	UDGL
51	DB08105	N-[(6-butoxynaphthalen-2-yl)sulfonyl]-L-glutamic acid	UDGL
52	DB01224	Quetiapine	DR

53	DB00311	Ethoxzolamide	CAII
54	DB01137	Levofloxacin	DNAG
55	DB08157	Ethyl 3-[4-(aminosulfonyl)phenyl]propanoate	CAII
56	DB07006	9-hydroxy-6-(3-hydroxypropyl)-4-(2-methoxyphenyl)pyrrolo[3,4-c]carbazole-1,3(2h,6h)-dione	W1LPK
57	DB00467	Enoxacin	DNAG
58	DB08107	N-{[6-(pentyloxy)naphthalen-2-yl]sulfonyl}-d-glutamic acid	UDGL
59	DB01194	Brinzolamide	CAII
60	DB09047	Finafloxacin	DNAG
61	DB00218	Moxifloxacin	DNAG
62	DB02861	4-(aminosulfonyl)-N-[(3,4,5-trifluorophenyl)methyl]-benzamide	CAII
63	DB07790	N-(2-methoxyethyl)-4-({4-[2-methyl-1-(1-methylethyl)-1h-imidazol-5-yl]pyrimidin-2-yl}amino)benzenesulfonamide	CDC2
64	DB07265	3-(9-hydroxy-1,3-dioxo-4-phenyl-2,3-dihydropyrrolo[3,4-c]carbazol-6(1h)-yl)propanoic acid	W1LPK
65	DB11491	Sarafloxacin	DNAG
66	DB08304	(3r)-3-cyclopentyl-7-[(4-methylpiperazin-1-yl)sulfonyl]-3,4-dihydro-2h-1,2-benzothiazine 1,1-dioxide	GR
67	DB05095	Cimicoxib	COX
68	DB08303	(3S)-3-cyclopentyl-6-methyl-7-[(4-methylpiperazin-1-yl)sulfonyl]-3,4-dihydro-2H-1,2,4-benzothiadiazine 1,1-dioxide	GR
69	DB06803	Niclosamide	RT
70	DB08729	5-ethoxy-4-(1-methyl-7-oxo-3-propyl-6,7-dihydro-1H-pyrazolo[4,3-d]pyrimidin-5-yl)thiophene-2-sulfonamide	CSCP
71	DB06891	5-{[(4-amino-3-chloro-5-fluorophenyl)sulfonyl]amino}-1,3,4-thiadiazole-2-sulfonamide	CAII
72	DB08701	2-(3-bromophenyl)-6-[(2-hydroxyethyl)amino]-1h-benzo[de]isoquinoline-1,3(2h)-dione	GP
73	DB07050	5-[(phenylsulfonyl)amino]-1,3,4-thiadiazole-2-sulfonamide	CAII
74	DB02429	4-(aminosulfonyl)-n-[(4-fluorophenyl)methyl]-benzamide	CAII
75	DB00998	Frovatriptan	SR
76	DB00685	Trovafloxacin	DNAG
77	DB08485	(1S,4S,5S)-1,4,5-trihydroxy-3-[3-(phenylthio)phenyl]cyclohex-2-ene-1-carboxylic acid	DD
78	DB01748	N-benzyl-4-sulfamoyl-benzamide	CAII
79	DB03307	4-[(6-amino-4-pyrimidinyl)amino]benzenesulfonamide	CDK2
80	DB03873	Tamibarotene	RAR
81	DB07683	N-(dibenzo[b,d]thiophen-3-ylsulfonyl)-L-valine	MM

Compounds identified by virtual screening of the DrugBank database are listed in rank order. The four compounds examined by pharmacological techniques in the Millar lab are highlighted in red.

* Target site of compounds identified in the DrugBank database. Abbreviations used:

AR: aldose reductase

CAII: carbonic anhydrase II

CAXIII: carbonic anhydrase XIII

CDK2: cyclin-dependent kinase 2

COX: cyclooxygenase

CSCP: cGMP-specific 3',5'-cyclic phosphodiesterase

DNAG: DNA gyrase/topoisomerase (target of fluoroquinolone antibiotics)

DD: 3-dehydroquinate dehydratase

DR: dopamine receptor

GP: genome polyprotein

GR: glutamate receptor

MM: macrophage metalloelastase

NKCC: $Na^+/K^+/2Cl^-$ cotransporter (target of loop diuretics)

PAO: primary-amine oxidase

PNMT: phenylethanolamine N-methyltransferase

RAR: retinoic acid receptor

RT: regulation of transcription

SR: serotonin receptor

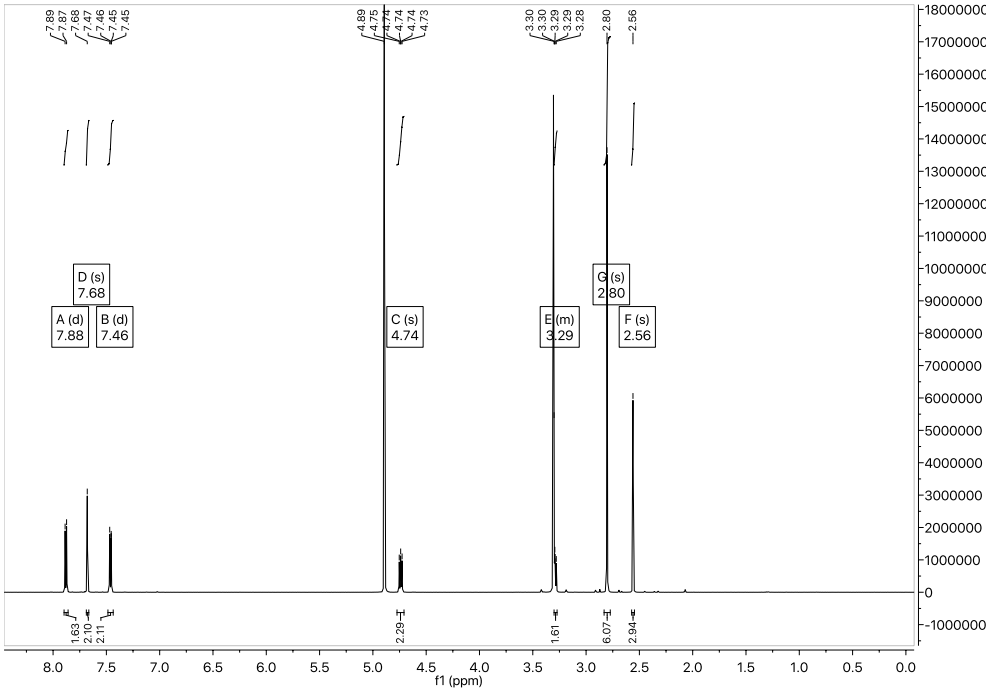
UDGL: UDP-N-acetylmuramoylalanine-D-glutamate ligase

W1LPK: Wee1-like protein kinase

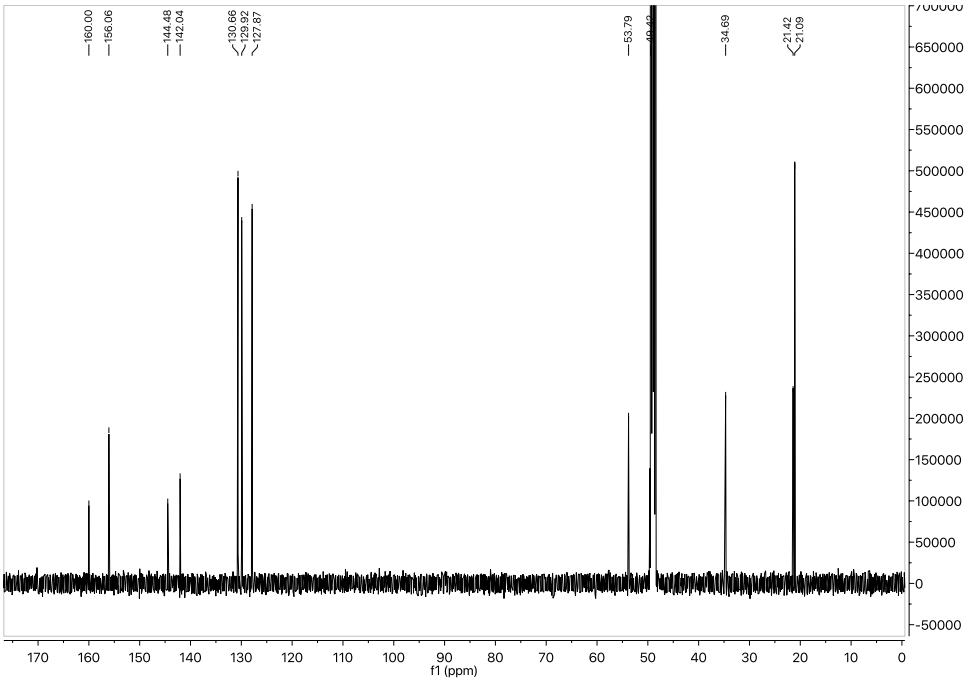
1.3.4 Chapter 5 – Appendix 4: Chemical data

1.3.4.1 2,4,6-trimethyl-1-(4-sulfamoylphenethyl)pyridin-1-ium tetrafluoroborate (DB04763)

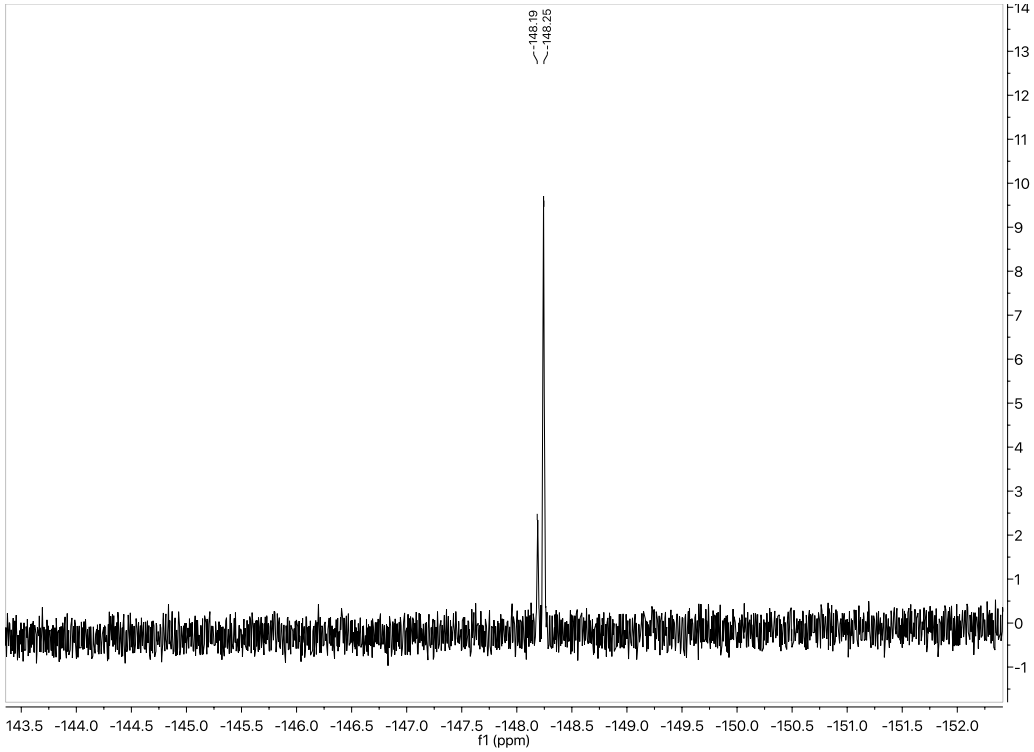
1.3.4.1.1 ^{1}H NMR



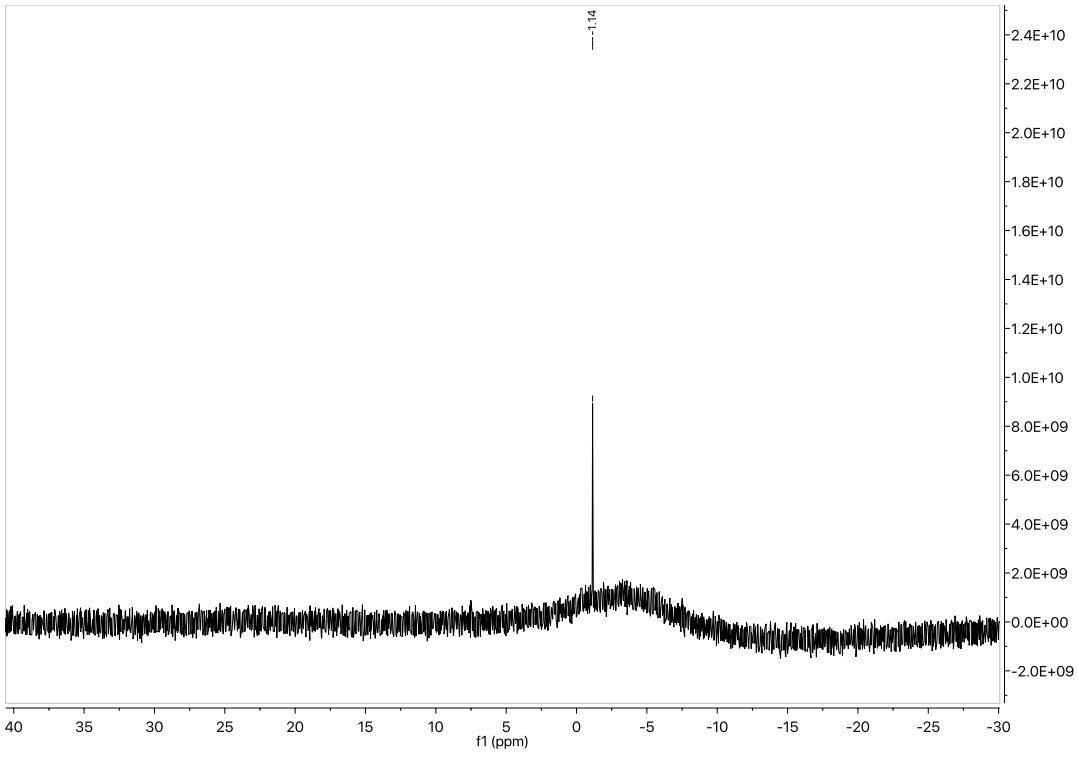
1.3.4.1.2 ^{13}C NMR



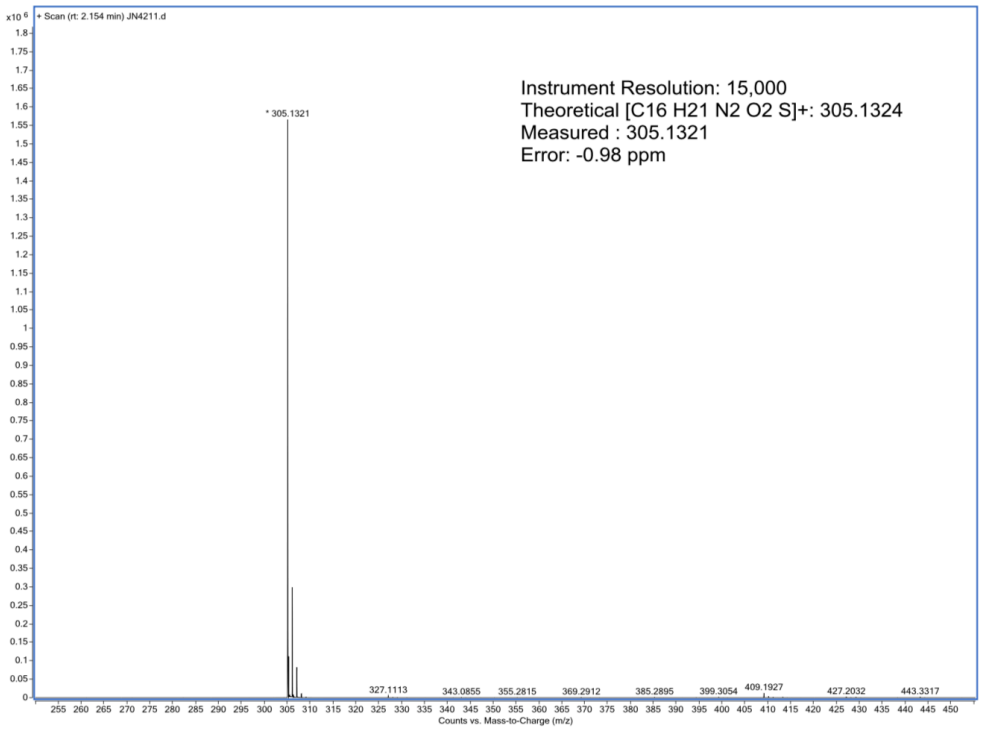
1.3.4.1.3 ^{19}F NMR



1.3.4.1.4 ^{11}B NMR

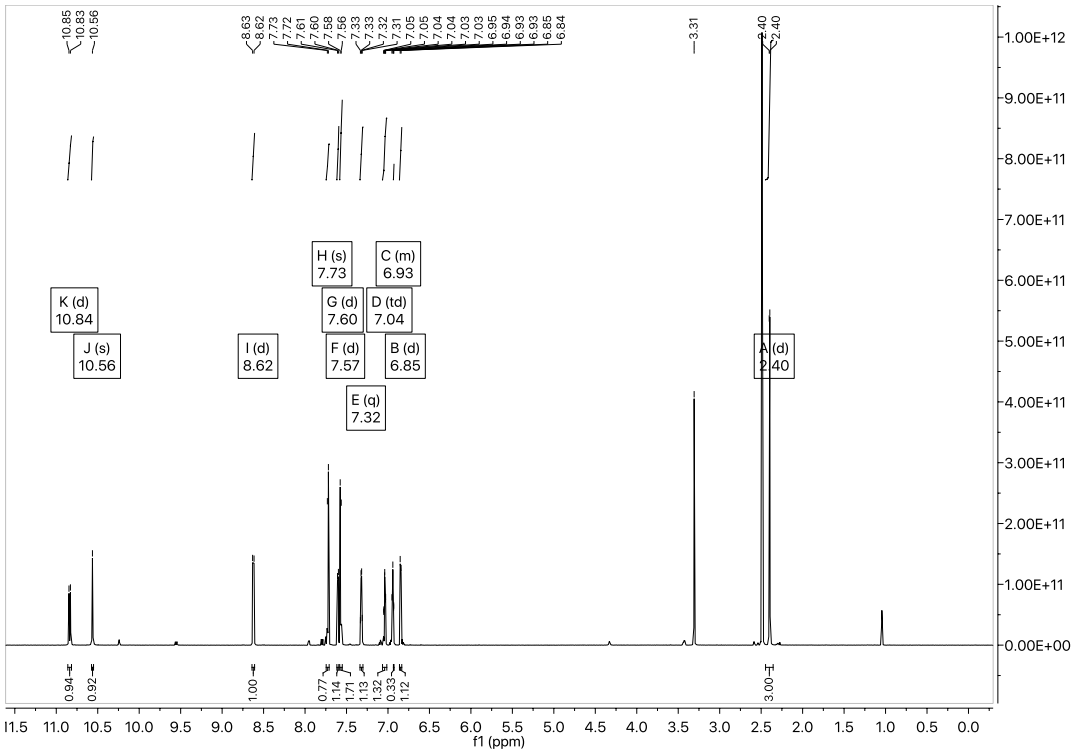


1.3.4.1.5 HRMS

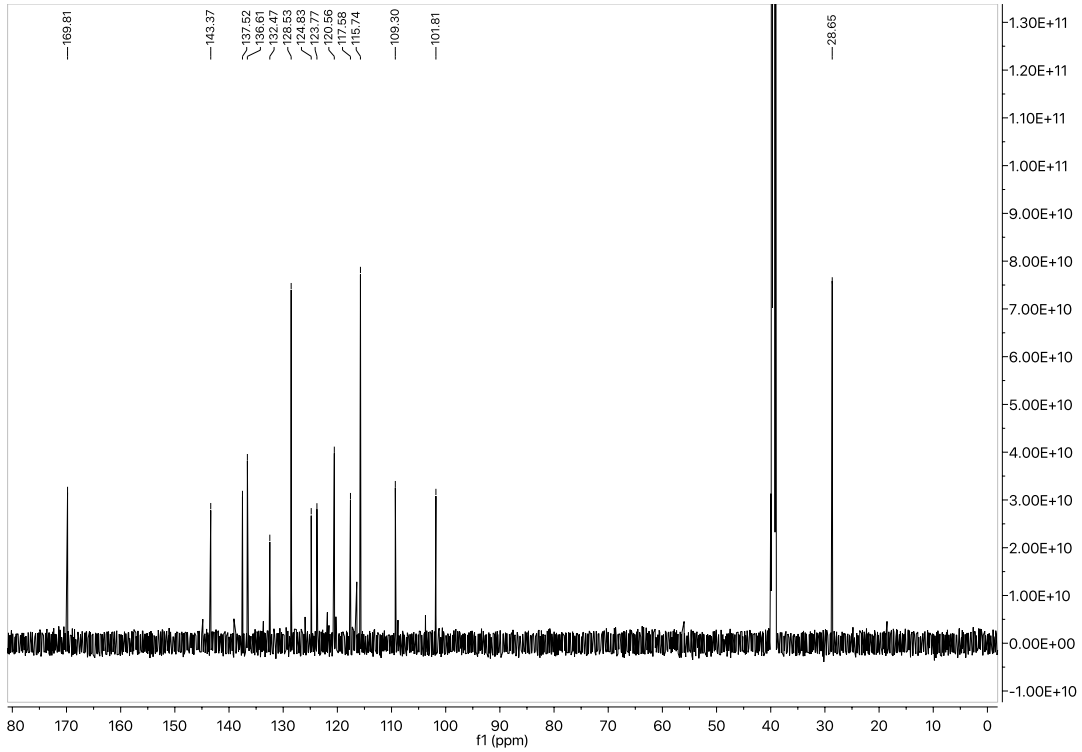


1.3.4.2 (Z)-N-methyl-4-(((2-oxoindolin-3-ylidene)methyl)amino)benzenesulfonamide (DB08122)

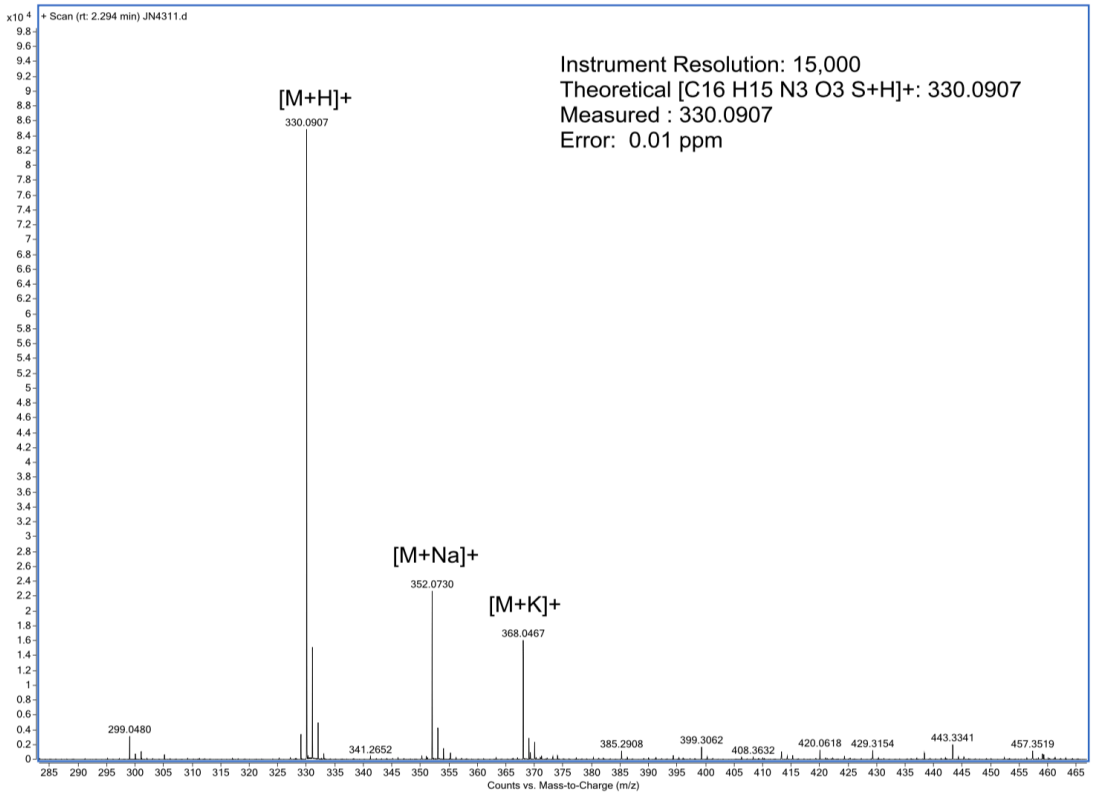
1.3.4.2.1 ^{1}H NMR



1.3.4.2.2 ^{13}C NMR



1.3.4.2.3 HRMS



1.4 Appendix – Chapter 6

1.4.1 Chapter 6 – Appendix 1: Sequence Alignment for use with *MODELLER*

Standard .pir format *MODELLER* sequence alignment (for explanation of sequence headers, see https://salilab.org/modeller/documentation.html). '-' represents a sequence alignment gap, '/' represents a chain break, '.' Represents a 'BLK' residue these are used for heteroatom assignments for ligands (in this case either glycosylation or sodium ions), 'w' represents water molecules. All amino acids are assigned with their standard 1 letter codes. In the 5-HT_{3A} model, all heteroatoms are assigned at the end of the PDB file, while in the re-ordered α4β2 nAChR model all heteroatoms are assigned at the end of each chain to which they belong.

```
>P1;6BE1_5ht3_TM4_HOH_NA
structureX:6BE1_5ht3_TM4_HOH_NA:397:A:602:E::::
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
-----------------------LAVRGLLQELSSIRHFLEKRDEMREVARDWLRVGYVL
DRLLFRIYLLAVLAYSITLVTLWSIWHYS---------------------/
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
-----------------------LAVRGLLQELSSIRHFLEKRDEMREVARDWLRVGYVL
DRLLFRIYLLAVLAYSITLVTLWSIWHYS---------------------/
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
-----------------------LAVRGLLQELSSIRHFLEKRDEMREVARDWLRVGYVL
DRLLFRIYLLAVLAYSITLVTLWSIWHYS---------------------/
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
-----------------------LAVRGLLQELSSIRHFLEKRDEMREVARDWLRVGYVL
DRLLFRIYLLAVLAYSITLVTLWSIWHYS---------------------/
```

```
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
----------------------LAVRGLLQELSSIRHFLEKRDEMREVARDWLRVGYVL
DRLLFRIYLLAVLAYSITLVTLWSIWHYS---------/./wwwwwwwwww-
*
>P1;6CNJ_a4b2_NAG_reorder
StructureY:6CNJ_a4b2_NAG_reorder:5:A:501:E::::
--------------------ETRAHAEERLLKKLF--SGYNKWSRPVANISDVVLVRFG
LSIAQLIDVDEKNQMMTTNVWVKQEWHDYKLRWDPADYENVTSIRIPSELIWRPDIVLYN
NADGDFAVTHLTKAHLFHDGRVQWTPPAIYKSSCSIDVTFFPFDQQNCTMKFGSWTYDKA
KIDLVNMH------------SRVDQLDFWESGEWVIVDAVGTYNTRKYECCAEI-YPDIT
YAFVIRRLPLFYTINLIIPCLLISCLTVLVFYLPSECGE-KITLCISVLLSLTVFLLLIT
EIIPSTSLVIPLIGEYLLFTMIFVTLSIVITVFVLNVHHRSPRTHTMPTWVRRVFLDIVP
RLLLMKR-----/----------------------------------------------
-------------------------------------------FERSVKEDWKYVAMVI
DRIFLWMFIIVCLLGTVGL---FLPPW----------.------------/
-------------------------DTEERLVEHLLDPSRYNKLIRPATNGSELVTVQLM
VSLAQLISVHEREQIMTTNVWLTQEWEDYRLTWKPEEFDNMKKVRLPSKHIWLPDVVLYN
NADGMYEVSFYSNAVVSYDGSIFWLPPAIYKSACKIEVKHFPFDQQNCTMKFRSWTYDRT
EIDLVLKS------------EVASLDDFTPSGEWDIVALPGRRNEN----PDDSTYVDIT
YDFIIRRKPLFYTINLIIPCVLITSLAILVFYLPSDCGE-KMTLCISVLLALTVFLLLIS
KIVPPTSLDVPLVGKYLMFTMVLVTFSIVTSVCVLNVHHRSPTTHTMAPWVKVVFLEKLP
ALLFMQQ-----/----------------------------------------------
-----------------------------------------------SVSEDWKYVAMVI
DRLFLWIFVFVCVFGTIGM---F--------------.------------/
-------------------------DTEERLVEHLLDPSRYNKLIRPATNGSELVTVQLM
VSLAQLISVHEREQIMTTNVWLTQEWEDYRLTWKPEEFDNMKKVRLPSKHIWLPDVVLYN
NADGMYEVSFYSNAVVSYDGSIFWLPPAIYKSACKIEVKHFPFDQQNCTMKFRSWTYDRT
EIDLVLKS------------EVASLDDFTPSGEWDIVALPGRRNEN----PDDSTYVDIT
YDFIIRRKPLFYTINLIIPCVLITSLAILVFYLPSDCGE-KMTLCISVLLALTVFLLLIS
KIVPPTSLDVPLVGKYLMFTMVLVTFSIVTSVCVLNVHHRSPTTHTMAPWVKVVFLEKLP
ALLFMQQ-----/----------------------------------------------
-----------------------------------------------SVSEDWKYVAMVI
DRLFLWIFVFVCVFGTIGM---F--------------.------------/
--------------------ETRAHAEERLLKKLF--SGYNKWSRPVANISDVVLVRFG
LSIAQLIDVDEKNQMMTTNVWVKQEWHDYKLRWDPADYENVTSIRIPSELIWRPDIVLYN
NADGDFAVTHLTKAHLFHDGRVQWTPPAIYKSSCSIDVTFFPFDQQNCTMKFGSWTYDKA
KIDLVNMH------------SRVDQLDFWESGEWVIVDAVGTYNTRKYECCAEI-YPDIT
YAFVIRRLPLFYTINLIIPCLLISCLTVLVFYLPSECGE-KITLCISVLLSLTVFLLLIT
EIIPSTSLVIPLIGEYLLFTMIFVTLSIVITVFVLNVHHRSPRTHTMPTWVRRVFLDIVP
RLLLMKR-----/----------------------------------------------
-------------------------------------------FERSVKEDWKYVAMVI
DRIFLWMFIIVCLLGTVGL---FLPPW----------.------------/
-------------------------DTEERLVEHLLDPSRYNKLIRPATNGSELVTVQLM
VSLAQLISVHEREQIMTTNVWLTQEWEDYRLTWKPEEFDNMKKVRLPSKHIWLPDVVLYN
NADGMYEVSFYSNAVVSYDGSIFWLPPAIYKSACKIEVKHFPFDQQNCTMKFRSWTYDRT
EIDLVLKS------------EVASLDDFTPSGEWDIVALPGRRNEN----PDDSTYVDIT
YDFIIRRKPLFYTINLIIPCVLITSLAILVFYLPSDCGE-KMTLCISVLLALTVFLLLIS
KIVPPTSLDVPLVGKYLMFTMVLVTFSIVTSVCVLNVHHRSPTTHTMAPWVKVVFLEKLP
ALLFMQQ-----/----------------------------------------------
-----------------------------------------------SVSEDWKYVAMVI
DRLFLWIFVFVCVFGTIGM---F--------------.--------------
*
```

```
>P1;2BG9_TM4CTER
StructureZ:2BG9_TM4CTER:407:A:476:E::::
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
DHILLCVFMLICIIGTVSV---FAGRLIELSQEG----------------/
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
-RLFLYIFITMCSIGTFSI---FLDASHNVPPDNPFA-------------/
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
DRLSMFIITPVMVLGTIFI---FVMGNFNRPPAK----------------/
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
DHILLCVFMLICIIGTVSV---FAGRLIELSQEG----------------/
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
DKACFWIALLLFSLGTLAI---FLTGHLNQVPE-------------------
*

>P1;ACHR_TORMA
Sequence:ACHR_TORMA:::::::
-----------------------SEHETRLVANLL--ENYNKVIRPVEHHTHFVDITVG
LQLIQLINVDEVNQIVETNVRLRQQWIDVRLRWNPADYGGIKKIRLPSDDVWLPDLVLYN
NADGDFAIVHMTKLLLDYTGKIMWTPPAIFKSYCEIIVTHFPFDQQNCTMKLGIWTYDGT
KVSISPES------------DRPDLSTFMESGEWVMKDYRGWKHWVYYTCCPDTPYLDIT
YHFIMQRIPLYFVVNVIIPCLLFSFLTVLVFYLPTDSGE-KMTLSISVLLSLTVFLLVIV
ELIPSTSSAVPLIGKYMLFTMIFVISSIIVTVVVINTHHRSPSTHTMPQWVRKIFINTIP
```

```
NVM---------/-------------------------------------------------
--------------------------SAIEGVKYIAEHMKSDEESSNAAEEWKYVAMVI
DHILLCVFMLICIIGTVSV---FAGRLIELSQEG---.------------/
-----------------------SVMEDTLLSVLF--ENYNPKVRPSQTVGDKVTVRVG
LTLTSLLILNEKNEEMTTSVFLNLAWTDYRLQWDPAAYEGIKDLSIPSDDVWQPDIVLMN
NNDGSFEITLHVNVLVQHTGAVSWHPSAIYRSSCTIKVMYFPFDWQNCTMVFKSYTYDTS
EVILQHALDA--/----VKEIMINQDAFTENGQWSIEHKPSRKNWRS----DDPSYEDVT
FYLIIQRKPLFYIVYTIVPCILISILAILVFYLPPDAGE-KMSLSISALLALTVFLLLLA
DKVPETSLSVPIIISYLMFIMILVAFSVILSVVVLNLHHRSPNTHTMPNWIRQIFIETLP
PFL---------/-------------------------------------------------
--------------------------EAVEAIKYIAEQLESASEFDDLKKDWQYVAMVA
DRLFLYIFITMCSIGTFSI---FLDASHNVPPDNPFA.------------/
-----------------------VNEEERLINDLLIVNKYNKHVRPVKHNNEVVNIALS
LTLSNLISLKETDETLTTNVWMDHAWYDHRLTWNASEYSDISILRLRPELIWIPDIVLQN
NNDGQYNVAYFCNVLVRPNGYVTWLPPAIFRSSCPINVLYFPFDWQNCSLKFTALNYNAN
EISMDLMT----/----IEWIIIDPEAFTENGEWEIIHKPAKKNIYGDKFPNGTNYQDVT
FYLIIRRKPLFYVINFITPCVLISFLAALAFYLPAESGE-KMSTAICVLLAQAVFLLLTS
QRLPETALAVPLIGKYLMFIMSLVTGVVVNCGIVLNFHFRTPSTHVLSTRVKQIFLEKLP
RIL---------/-------------------------------------------------
--------------------------SGIDSTNYIVKQIKEKNAYDEEVGNWNLVGQTI
DRLSMFIITPVMVLGTIFI---FVMGNRPPAK-----.------------/
-----------------------SEHETRLVANLL--ENYNKVIRPVEHHTHFVDITVG
LQLIQLINVDEVNQIVETNVRLRQQWIDVRLRWNPADYGGIKKIRLPSDDVWLPDLVLYN
NADGDFAIVHMTKLLLDYTGKIMWTPPAIFKSYCEIIVTHFPFDQQNCTMKLGIWTYDGT
KVSISPES-----------DRPDLSTFMESGEWVMKDYRGWKHWVYYTCCPDTPYLDIT
YHFIMQRIPLYFVVNVIIPCLLFSFLTVLVFYLPTDSGE-KMTLSISVLLSLTVFLLVIV
ELIPSTSSAVPLIGKYMLFTMIFVISSIIVTVVVINTHHRSPSTHTMPQWVRKIFINTIP
NVM---------/-------------------------------------------------
--------------------------SAIEGVKYIAEHMKSDEESSNAAEEWKYVAMVI
DHILLCVFMLICIIGTVSV---FAGRLIELSQEG---.------------/
-----------------------NEEGRLIEKLL--GDYDKRIKPAKTLDHVIDVTLK
LTLTNLISLNEKEEALTTNVWIEIQWNDYRLSWNTSEYEGIDLVRIPSELLWLPDVVLEN
NVDGQFEVAYYANVLVYNDGSMYWLPPAIYRSTCPIAVTYFPFDWQNCSLVFRSQTYNAH
EVNLQLSAEEG-/----VEWIHIDPEDFTENGEWTIRHRPAKKNYNWQLTKDDIDFQEII
FFLIIQRKPLFYIINIIAPCVLISSLVVLVYFLPAQAGGQKCTLSISVLLAQTIFLFLIA
QKVPETSLNVPLIGKYLIFVMFVSLVIVTNCVIVLNVSLRTPNTHSLSEKIKHLFLEFLP
KYL---------/-------------------------------------------------
--------------------------SCVEACNFIAKSTKEQNDSGSENENWVLIGKVI
DKACFWIALLLFSLGTLAI---FLTGHLNQVPE----././wwwwwwwwww-
*
```

1.4.2 Chapter 6 – Appendix 2: *MODELLER* script

```
# Homology modeling by the automodel class
from modeller import *              # Load standard Modeller
classes
from modeller.automodel import *    # Load the automodel class

# Redefine the special_patches routine to include the additional
disulfides
# (this routine is empty by default):
class mymodel(automodel):
    def special_patches(self, aln):
      self.patch_ss_templates(aln)


log.verbose()    # request verbose output
```

```
env = environ(rand_seed=-556)  # create a new MODELLER
environment to build this model in

env.io.atom_files_directory = ['.', '../atom_files']
env.io.hetatm = True #read hetatms from pdb to generate ligands
in the model
env.io.water = True

a = mymodel(env,
            alnfile  = 'multi-chain_multi-seq.pir',     #
alignment filename
            knowns   =
('6CNJ_a4b2_NAG_reorder','6BE1_5ht3_TM4_HOH_NA','2BG9_TM4CTER'),
# codes of the templates
            sequence = 'ACHR_TORMA',         # code of the target
          assess_methods=assess.DOPE)      # gives DOPE energy
readouts at bottom of log file

a.starting_model= 1                 # index of the first model
a.ending_model  = 200               # index of the last model
                                    # (determines how many models
to calculate)

a.md_level = None                       # do not perform any
optimisation
a.make()                            # do the actual homology
modeling

# Get a list of all successfully built models from a.outputs
ok_models = filter(lambda x: x['failure'] is None, a.outputs)

# Rank the models by DOPE score
key = 'DOPE score'
ok_models.sort(lambda a,b: cmp(a[key], b[key]))

# Get top model
m = ok_models[0]
print "Top model: %s (DOPE score %.3f)" % (m['name'], m[key])
```