Generating HSUM configuration

(Work done in ticket AT4-1317)

The HSUM module uses a set of configuration registers to determine which FOP rows to use in harmonic summation. The module goes to address 'Address = 0x000000 + SUMMER_INSTANCE * 131072 + ANALYSIS_RUN * 65536 + SEED_NUM * 2048 + ACC_AMBIGUITY_NUM * 128 + HARMONIC * 8' to find the FOP row value to use for given seed, ambiguity slope, and harmonic.

As the track of the summing slopes changes from linear to non-linear in the FOP in the case of multiplicative filters, the values at the addresses in the HPSEL registers need to change. In the initially used uniform sampling of filters, the separation between filters was 5 drift bins. Using a 1 filter tab (filter tab=2*drift bins +1) as quanta, each seed was searched in 11 slopes (2*5+1). In the multiplicative sampling, the separation between filters is not constant and keeps changing with changing width. We can not use the concept of a quanta in this case. Instead, we use the possible error due to spacing in width at the seed and how this error will translate to the highest harmonic available. We then use all the filters within the error (at the highest available harmonic) to construct summing slopes. Using the propagation of error in higher harmonics can be generalized to any choice of filters.

Generating summing trees for a given seed, harmonic, and ambiguity slope:

Say we are working on a seed in filter P(i) and the filter widths of the neighboring filters are P(i-1) and P(i+1). If we had a continuous FOP plane (i.e. mo quantization in filter width or frequency), then each seed would be associated with a unique summing path. While generating the summing tree for P(i) we will consider nine possible seeds within P(i-1) to P(i).

1. First, compute the harmonic locations (summing branch) for [P(i-1)+P(i)]/2 for the current filter choices.
2. Use this branch as a benchmark to generate the branches for the other eight seeds within P(i-1) and P(i).
3. We use [1,3,3,5,5,7,7,9] neighboring filters around the benchmark summing branch for harmonics [1,2,3,4,5,6,7,8] to generate the 8 other summing branches.

Martin has made some changes in the HSUM configuration to accommodate two Summer trees and two HSUM runs. The two summer trees can run simultaneously and search in positive and negative acceleration respectively. A single HSUM run with two summer trees can search in 21 seed locations with positive acceleration and 21 seed locations with negative acceleration. We plan to use two HSUM runs with two summer trees to cover seeds in all 85 FOP rows.

I have created two Python scripts for the two HSUM runs to generate summing slopes for FOP seeds. The scripts create slopes for seeds and provide FOP row value for all available harmonics (<8) for each slope. The results are saved in two Excel sheets, one for each of the two summer trees (positive and negative acceleration). The first script for HSUM run 1 creates the HPSEL table for seeds 1 to 21 (summer tree 1) and -1 to -21 (summer tree 2), and the second script for HSUM run 2 creates the HPSEL table for seeds 21 to 42 (summer tree 1) and -21 to -42 (summer tree 2).

Martin’s 5-slope configuration to complete the search in one run:

The number of branches in the summing tree reduces when we go to seeds in wider filters as many high harmonics of these seeds fall off the FOP. In the new multiplicative sampling of filter widths, only a single branch is required and the filter width scales multiplicatively and spacing covers the propagation of error in the higher harmonics. However, due to spectral resolution, narrower filters can not be multiplicatively spaced. In the current settings, the first 15 filters are uniformly spaced and then the spacing becomes multiplicative. So, only one branch is used for seeds wider than 15 filters. So, we need 9 slopes in P[0] and P[1], and after this number of required slopes drastically reduces. In the first run where we are covering the first 21 positive and negative acceleration filters, we are using a total of 72 slopes per summer instance (for all seeds in this run combined). Since the two summer instances run truly simultaneously we need not worry about the second summer instance. Now, in run 2 where we cover the last 20 filters (widest ones), we are only using 20 slopes. In run1 and run2 combined, we are using only 92 independent summings out of (21*11=)231 available summings.

Martin came up with the idea that if we arranged all the independent summing (i.e. slopes) in a specific manner, we could complete all these sums using only a single run. There is a global parameter that decides how many slopes to use for each seed. We have 92 slopes for 21 seeds, hence if we use 5 slopes per seed then we will complete all 92 slopes in a single run. So, we decided to arrange the 92 sums in the first five slopes of 21 seeds of a single run and set the global parameter of the number of slopes to 5.

Generating final configuration:

The HSUM module requires only a set of register addresses and filter values (i.e. basically row numbers in FOP) so that it can store all these values in respective registers and use them during the HSUM run.
We also add a header to this list of addresses and values describing the global settings of HSUM.

How to use the scripts:

We have four scripts:

1. Complete_HPSEL_run1
2. Complete_HPSEL_run2
3. Complete_HPSEL_combine
4. Complete_final_HPSEL_5slopes

The first two scripts are the scripts that create all the independent summings (i.e. slopes for all seeds) for run1 and run2. These scripts require the filter widths, transition filter, and values (filter values to be stored in HPSEL registers). The transition filter is the filter width where we transition from uniform to multiplicative sampling. For currently used filters 15 is the transition filter.

These scripts write out multiple Excel sheets with various information about the summing slopes.

The third script (Complete_HPSEL_combine) combines the two HSUM runs generated by the first two scripts into a single run according to the 5 slope strategy as discussed above.

The last script (Complete_final_HPSEL_5slopes) creates the final list of addresses and values along with global configuration. It writes out three files.

Complete_HPSEL_final_addresses_values_single_run_5slopes.xlsx : This excel sheet has detailed information about the final HPSEL table

Complete_HPSEL_only_addresses_values_single_run_5slopes.txt : This is a text file only with the list of addresses and values

Complete_HPSEL_only_addresses_values_single_run_5slopes_with_config.txt : This text file has a global configuration along with the list.

Testing the new configurations:

Martin tested the newly generated configurations with the FDAS. The configurations are working fine. We get detections of most of the injected candidates.

Martin also created comparisons with the previously used uniform filters. In the comparison, he spotted a problem with the multiplicative filters. We are facing a huge degradation in recovered power for some of the very wide filters. When the actual drift in spectra and the nearest convolution filter width differ by more than 5 drift bins, the sensitivity drops by a factor of 2-3.

One example of this is the following:

One test spectra was injected with a 500 Hz pulsar with an acceleration of 280 m/s^2. This translates to 133 bin drifts in the fundamental harmonic. The uniform filters detected this with a power of 226. However the multiplicative filters could find only a power of 82. The mismatch between actual drift and filter width is 2 (nearest width 135) in the uniform sampling while it is 6 (nearest filter 139) in the multiplicative sampling of widths.

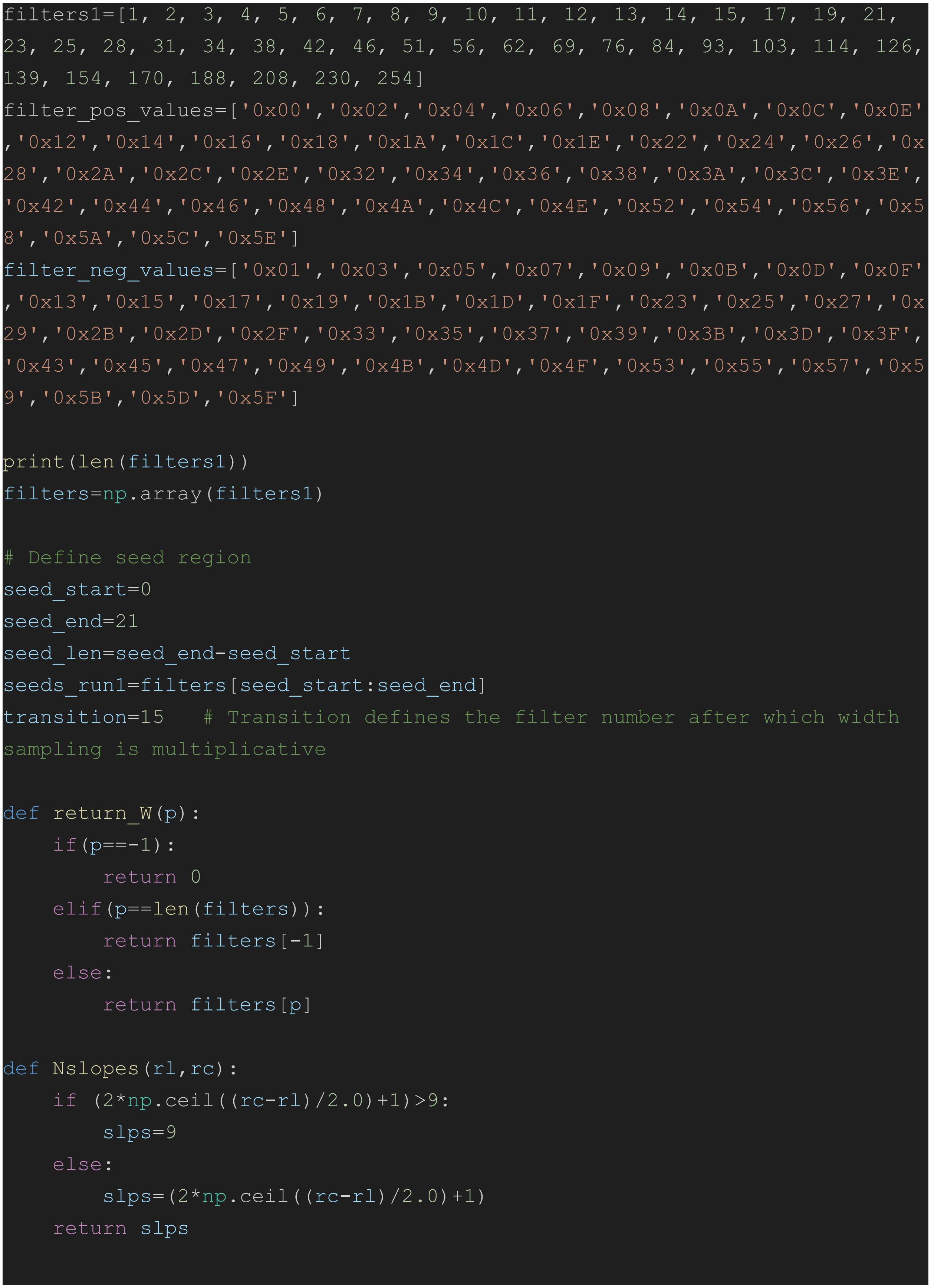
If we consider a boxcar in place of complex filters, the mismatch of 6 bins for a filter of width 139 will create a power loss of 2-3% but we are seeing a 3 times reduction in power.

We need to understand the reason behind this huge degradation in recovered power. Maybe it has something to do with mismatch in phases as well.

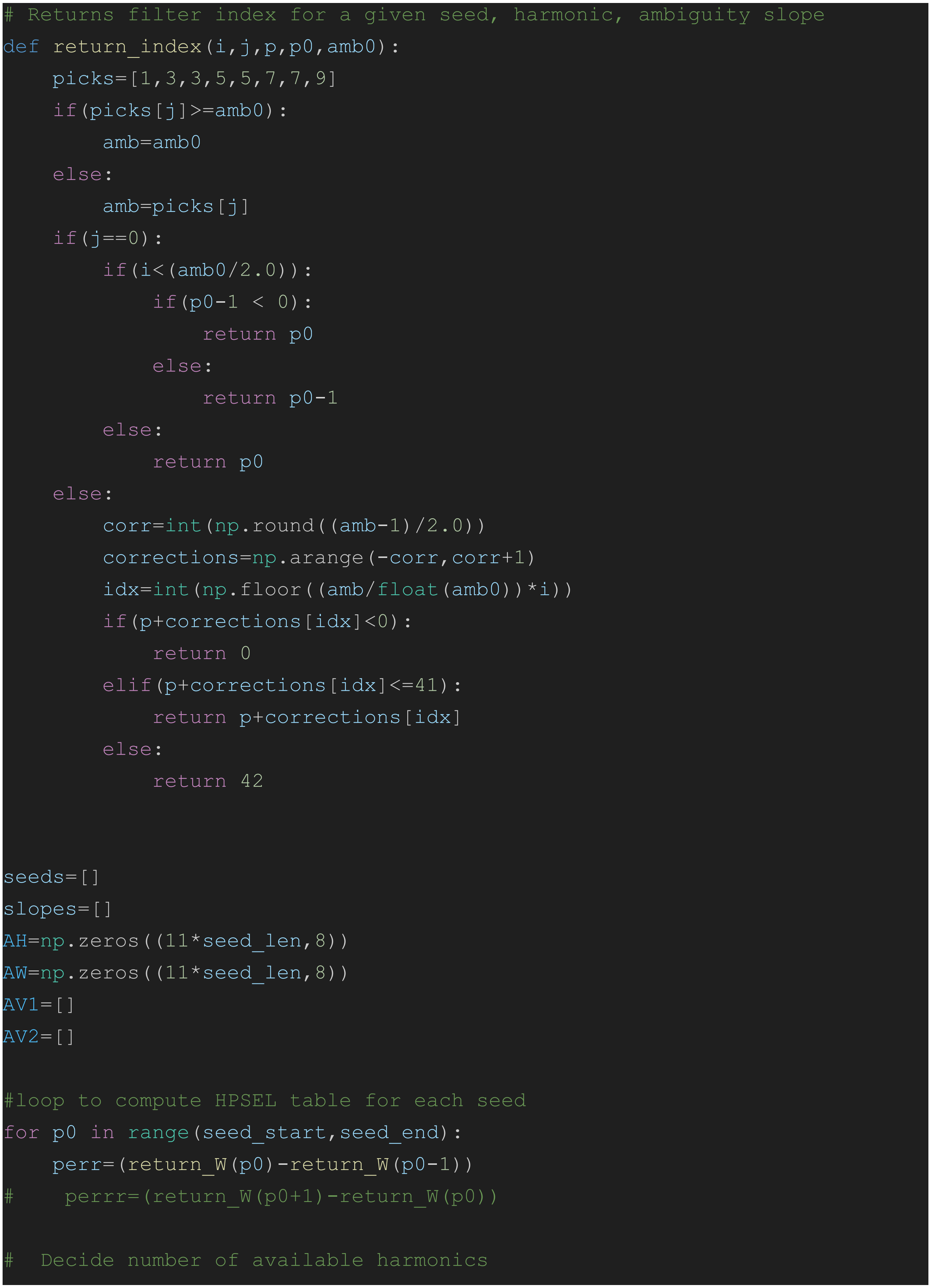
Scripts:

1. Complete_HPSEL_run1

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

filters2=[]           # Filter 2 is uniformly sampled filters

for i in range(1,43):
    filters2.append(i*5)

#   Filters is the new multiplicatively sampled filters
```



```
filters1=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 19, 21, 23, 25, 28, 31, 34, 38, 42, 46, 51, 56, 62, 69, 76, 84, 93, 103, 114, 126, 139, 154, 170, 188, 208, 230, 254]
filter_pos_values=['0x00','0x02','0x04','0x06','0x08','0x0A','0x0C','0x0E','0x12','0x14','0x16','0x18','0x1A','0x1C','0x1E','0x22','0x24','0x26','0x28','0x2A','0x2C','0x2E','0x32','0x34','0x36','0x38','0x3A','0x3C','0x3E','0x42','0x44','0x46','0x48','0x4A','0x4C','0x4E','0x52','0x54','0x56','0x58','0x5A','0x5C','0x5E']
filter_neg_values=['0x01','0x03','0x05','0x07','0x09','0x0B','0x0D','0x0F','0x13','0x15','0x17','0x19','0x1B','0x1D','0x1F','0x23','0x25','0x27','0x29','0x2B','0x2D','0x2F','0x33','0x35','0x37','0x39','0x3B','0x3D','0x3F','0x43','0x45','0x47','0x49','0x4B','0x4D','0x4F','0x53','0x55','0x57','0x59','0x5B','0x5D','0x5F']

print(len(filters1))
filters=np.array(filters1)

# Define seed region
seed_start=0
seed_end=21
seed_len=seed_end-seed_start
seeds_run1=filters[seed_start:seed_end]
transition=15   # Transition defines the filter number after which width sampling is multiplicative

def return_W(p):
    if(p==-1):
        return 0
    elif(p==len(filters)):
        return filters[-1]
    else:
        return filters[p]

def Nslopes(rl,rc):
    if (2*np.ceil((rc-rl)/2.0)+1)>9:
        slps=9
    else:
        slps=(2*np.ceil((rc-rl)/2.0)+1)
    return slps
```



```
# Returns filter index for a given seed, harmonic, ambiguity slope
def return_index(i,j,p,p0,amb0):
    picks=[1,3,3,5,5,7,7,9]
    if(picks[j]>=amb0):
        amb=amb0
    else:
        amb=picks[j]
    if(j==0):
        if(i<(amb0/2.0)):
            if(p0-1 < 0):
                return p0
            else:
                return p0-1
        else:
            return p0
    else:
        corr=int(np.round((amb-1)/2.0))
        corrections=np.arange(-corr,corr+1)
        idx=int(np.floor((amb/float(amb0))*i))
        if(p+corrections[idx]<0):
            return 0
        elif(p+corrections[idx]<=41):
            return p+corrections[idx]
        else:
            return 42


seeds=[]
slopes=[]
AH=np.zeros((11*seed_len,8))
AW=np.zeros((11*seed_len,8))
AV1=[]
AV2=[]

#loop to compute HPSEL table for each seed
for p0 in range(seed_start,seed_end):
    perr=(return_W(p0)-return_W(p0-1))
#    perrr=(return_W(p0+1)-return_W(p0))

#  Decide number of available harmonics
```

```
    if(8*seeds_run1[p0-seed_start] <= filters[-1]):
        harms=8
    else:
        for i in range(8):
            if((i+1)*seeds_run1[p0-seed_start] > filters[-1]):
                harms=i
                break

#  Decide number of ambiguity slopes
    if(p0>transition/2.0):
        if(p0<transition):
            amb_slopes=3
        else:
            amb_slopes=1
    else:
        th=int(float(transition)/filters[p0])
        Rl=np.argmin(np.abs(filters-((th*(filters[p0]-perr)))))
#        Rr=np.argmin(np.abs(filters-((th*(filters[p0]+perr)))))
        Rc=np.argmin(np.abs(filters-((th*(filters[p0])))))
        amb_slopes=Nslopes(Rl,Rc)
        print(amb_slopes)

#  Work out filters for each harmonic of this seed
    filts=[]
    for j in range(1,harms+1):
        ph=np.argmin(np.abs(filters-((j*(filters[p0]-perr/2.0)))))
        filts.append(ph)


    for i in range(0,11):
        seeds.append(p0)
        slopes.append(i+1)
        V1=[]
        V2=[]
        for j in range(0,8):
            if(i+1>amb_slopes):
                AH[(p0-seed_start)*11+i,j]=np.NAN
                AW[(p0-seed_start)*11+i,j]=np.NAN
                V1.append('0x60')
                V2.append('0x60')
            elif(j+1>harms):
                AH[(p0-seed_start)*11+i,j]=np.NAN
                AW[(p0-seed_start)*11+i,j]=np.NAN
                V1.append('0x60')
                V2.append('0x60')
            else:

AH[(p0-seed_start)*11+i,j]=return_index(i,j,filts[j],p0,amb_slopes)

AW[(p0-seed_start)*11+i,j]=filters[return_index(i,j,filts[j],p0,amb_slopes
)]

V1.append(filter_pos_values[return_index(i,j,filts[j],p0,amb_slopes)])

V2.append(filter_neg_values[return_index(i,j,filts[j],p0,amb_slopes)])
        AV1.append(V1)
        AV2.append(V2)

H1=AH[:,0]
H2=AH[:,1]
H3=AH[:,2]
H4=AH[:,3]
H5=AH[:,4]
H6=AH[:,5]
H7=AH[:,6]
H8=AH[:,7]
dataframe=pd.DataFrame({'seeds':seeds, 'slopes':slopes, 'H1':H1, 'H2':H2,
'H3':H3, 'H4':H4, 'H5':H5, 'H6':H6, 'H7':H7, 'H8':H8})
dataframe.to_excel('Complete_HPSEL_run1_seed_slope_harmonic_number.xlsx')
seeds1=seeds
seeds=filters[seeds]

H1=AW[:,0]
H2=AW[:,1]
H3=AW[:,2]
H4=AW[:,3]
H5=AW[:,4]
H6=AW[:,5]
H7=AW[:,6]
```

```
H8=AW[:,7]
dataframe=pd.DataFrame({'seeds':seeds, 'slopes':slopes, 'H1':H1, 'H2':H2, 'H3':H3, 'H4':H4, 'H5':H5, 'H6':H6, 'H7':H7, 'H8':H8})
dataframe.to_excel('Complete_HPSEL_run1_seed_slope_harmonic_width.xlsx')

H1=[]
H2=[]
H3=[]
H4=[]
H5=[]
H6=[]
H7=[]
H8=[]
for x in AV1:
    H1.append(x[0])
    H2.append(x[1])
    H3.append(x[2])
    H4.append(x[3])
    H5.append(x[4])
    H6.append(x[5])
    H7.append(x[6])
    H8.append(x[7])
dataframe=pd.DataFrame({'seeds':seeds1, 'slopes':slopes, 'H1':H1, 'H2':H2, 'H3':H3, 'H4':H4, 'H5':H5, 'H6':H6, 'H7':H7, 'H8':H8})
dataframe.to_excel('Complete_HPSEL_run1_seed_slope_harmonic_values_summer1.xlsx')

H1=[]
H2=[]
H3=[]
H4=[]
H5=[]
H6=[]
H7=[]
H8=[]
for x in AV2:
    H1.append(x[0])
    H2.append(x[1])
    H3.append(x[2])
    H4.append(x[3])
    H5.append(x[4])
    H6.append(x[5])
    H7.append(x[6])
    H8.append(x[7])
dataframe=pd.DataFrame({'seeds':seeds1, 'slopes':slopes, 'H1':H1, 'H2':H2, 'H3':H3, 'H4':H4, 'H5':H5, 'H6':H6, 'H7':H7, 'H8':H8})
dataframe.to_excel('Complete_HPSEL_run1_seed_slope_harmonic_values_summer2.xlsx')
```

2. Complete_HPSEL_run2

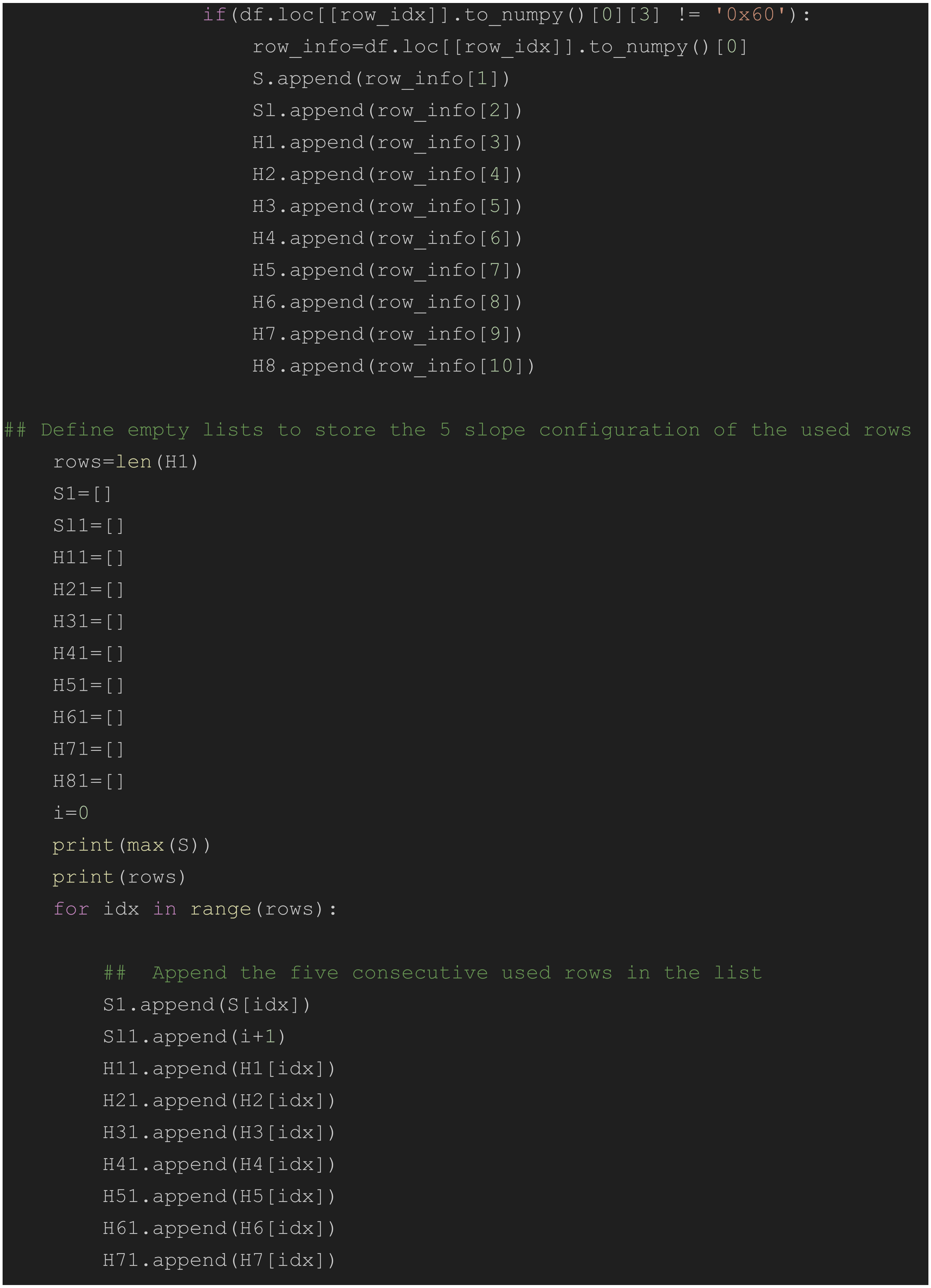
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

filters2=[]           # Filter 2 is uniformly sampled filters

for i in range(1,43):
    filters2.append(i*5)

#   Filters is the new multiplicatively sampled filters
filters1=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 19, 21, 23, 25, 28, 31, 34, 38, 42, 46, 51, 56, 62, 69, 76, 84, 93, 103, 114, 126, 139, 154, 170, 188, 208, 230, 254]
filter_pos_values=['0x00','0x02','0x04','0x06','0x08','0x0A','0x0C','0x0E','0x12','0x14','0x16','0x18','0x1A','0x1C','0x1E','0x22','0x24','0x26','0x28','0x2A','0x2C','0x2E','0x32','0x34','0x36','0x38','0x3A','0x3C','0x3E','0x42','0x44','0x46','0x48','0x4A','0x4C','0x4E','0x52','0x54','0x56','0x58','0x5A','0x5C','0x5E']
filter_neg_values=['0x01','0x03','0x05','0x07','0x09','0x0B','0x0D','0x0F','0x13','0x15','0x17','0x19','0x1B','0x1D','0x1F','0x23','0x25','0x27','0x29','0x2B','0x2D','0x2F','0x33','0x35','0x37','0x39','0x3B','0x3D','0x3F','0x43','0x45','0x47','0x49','0x4B','0x4D','0x4F','0x53','0x55','0x57','0x59','0x5B','0x5D','0x5F']

print(len(filters1))
```

```
filters=np.array(filters1)

# Define seed region
seed_start=21
seed_end=42
seed_len=seed_end-seed_start
seeds_run1=filters[seed_start:seed_end]
transition=15   # Transition defines the filter number after which width sampling is multiplicative

def return_W(p):
    if(p==-1):
        return 0
    elif(p==len(filters)):
        return filters[-1]
    else:
        return filters[p]

def Nslopes(rl,rc):
    if (2*np.ceil((rc-rl)/2.0)+1)>9:
        slps=9
    else:
        slps=(2*np.ceil((rc-rl)/2.0)+1)
    return slps


def return_index(i,j,p,ho,amb0):
    picks=[1,3,3,5,5,7,7,9]
    if(picks[j]>amb0):
        amb=amb0
    else:
        amb=picks[j]
    if(j==0):
       if(i<(amb0/2.0)):
            if(p0-1<0):
                return p0
            else:
                return p0-1
       else:
           return p0
    else:
        corr=int(np.round((amb-1)/2.0))
        corrections=np.arange(-corr,corr+1)
        idx=int(np.floor((amb/float(amb0))*i))
        if(p+corrections[idx]<0):
            return 0
        elif(p+corrections[idx]<=41):
            return p+corrections[idx]
        else:
            return 42


seeds=[]
slopes=[]
AH=np.zeros((11*seed_len,8))
AW=np.zeros((11*seed_len,8))
AV1=[]
AV2=[]
for p0 in range(seed_start,seed_end):
    perr=(return_W(p0)-return_W(p0-1))
#    perrr=(return_W(p0+1)-return_W(p0))

#  Decide number of available harmonics
    if(8*seeds_run1[p0-seed_start] <= filters[-1]):
        harms=8
    else:
        for i in range(8):
            if((i+1)*seeds_run1[p0-seed_start] > filters[-1]):
                harms=i
                break

#  Decide number of ambiguity slopes
    if(p0>transition/2.0):
        if(p0<transition):
            amb_slopes=3
        else:
            amb_slopes=1
    else:
        th=int(float(transition)/filters[p0])
        Rl=np.argmin(np.abs(filters-((th*(filters[p0]-perr)))))
```

```
#        Rr=np.argmin(np.abs(filters-((th*(filters[p0]+perrr)))))
        Rc=np.argmin(np.abs(filters-((th*(filters[p0])))))
        amb_slopes=Nslopes(Rl,Rc)
        print(amb_slopes)

#  Work out filters for each harmonic of this seed
    filts=[]
    for j in range(1,harms+1):
        ph=np.argmin(np.abs(filters-np.ceil((j*(filters[p0]-perr/2.0)))))
        filts.append(ph)


    for i in range(0,11):
        seeds.append(p0)
        slopes.append(i+1)
        V1=[]
        V2=[]
        for j in range(0,8):
            if(i+1>amb_slopes):
                AH[(p0-seed_start)*11+i,j]=np.NAN
                AW[(p0-seed_start)*11+i,j]=np.NAN
                V1.append('0x60')
                V2.append('0x60')

            elif(j+1>harms):
                AH[(p0-seed_start)*11+i,j]=np.NAN
                AW[(p0-seed_start)*11+i,j]=np.NAN
                V1.append('0x60')
                V2.append('0x60')
            else:

AH[(p0-seed_start)*11+i,j]=return_index(i,j,filts[j],harms,amb_slopes)

AW[(p0-seed_start)*11+i,j]=filters[return_index(i,j,filts[j],harms,amb_slo
pes)]

V1.append(filter_pos_values[return_index(i,j,filts[j],harms,amb_slopes)])

V2.append(filter_neg_values[return_index(i,j,filts[j],harms,amb_slopes)])
        AV1.append(V1)
        AV2.append(V2)

H1=AH[:,0]
H2=AH[:,1]
H3=AH[:,2]
H4=AH[:,3]
H5=AH[:,4]
H6=AH[:,5]
H7=AH[:,6]
H8=AH[:,7]
dataframe=pd.DataFrame({'seeds':seeds, 'slopes':slopes, 'H1':H1, 'H2':H2,
'H3':H3, 'H4':H4, 'H5':H5, 'H6':H6, 'H7':H7, 'H8':H8})
dataframe.to_excel('Complete_HPSEL_run2_seed_slope_harmonic_number.xlsx')
seeds1=seeds
seeds=filters[seeds]

H1=AW[:,0]
H2=AW[:,1]
H3=AW[:,2]
H4=AW[:,3]
H5=AW[:,4]
H6=AW[:,5]
H7=AW[:,6]
H8=AW[:,7]
dataframe=pd.DataFrame({'seeds':seeds, 'slopes':slopes, 'H1':H1, 'H2':H2,
'H3':H3, 'H4':H4, 'H5':H5, 'H6':H6, 'H7':H7, 'H8':H8})
dataframe.to_excel('Complete_HPSEL_run2_seed_slope_harmonic_width.xlsx')

H1=[]
H2=[]
H3=[]
H4=[]
H5=[]
H6=[]
H7=[]
H8=[]
for x in AV1:
    H1.append(x[0])
    H2.append(x[1])
    H3.append(x[2])
    H4.append(x[3])
    H5.append(x[4])
    H6.append(x[5])
    H7.append(x[6])
    H8.append(x[7])
dataframe=pd.DataFrame({'seeds':seeds1, 'slopes':slopes, 'H1':H1, 'H2':H2, 'H3':H3, 'H4':H4, 'H5':H5, 'H6':H6, 'H7':H7, 'H8':H8})
dataframe.to_excel('Complete_HPSEL_run2_seed_slope_harmonic_values_summer1.xlsx')

H1=[]
H2=[]
H3=[]
H4=[]
H5=[]
H6=[]
H7=[]
H8=[]
for x in AV2:
    H1.append(x[0])
    H2.append(x[1])
    H3.append(x[2])
    H4.append(x[3])
    H5.append(x[4])
    H6.append(x[5])
    H7.append(x[6])
    H8.append(x[7])
dataframe=pd.DataFrame({'seeds':seeds1, 'slopes':slopes, 'H1':H1, 'H2':H2, 'H3':H3, 'H4':H4, 'H5':H5, 'H6':H6, 'H7':H7, 'H8':H8})
dataframe.to_excel('Complete_HPSEL_run2_seed_slope_harmonic_values_summer2.xlsx')
```

3. Complete_HPSEL_combine

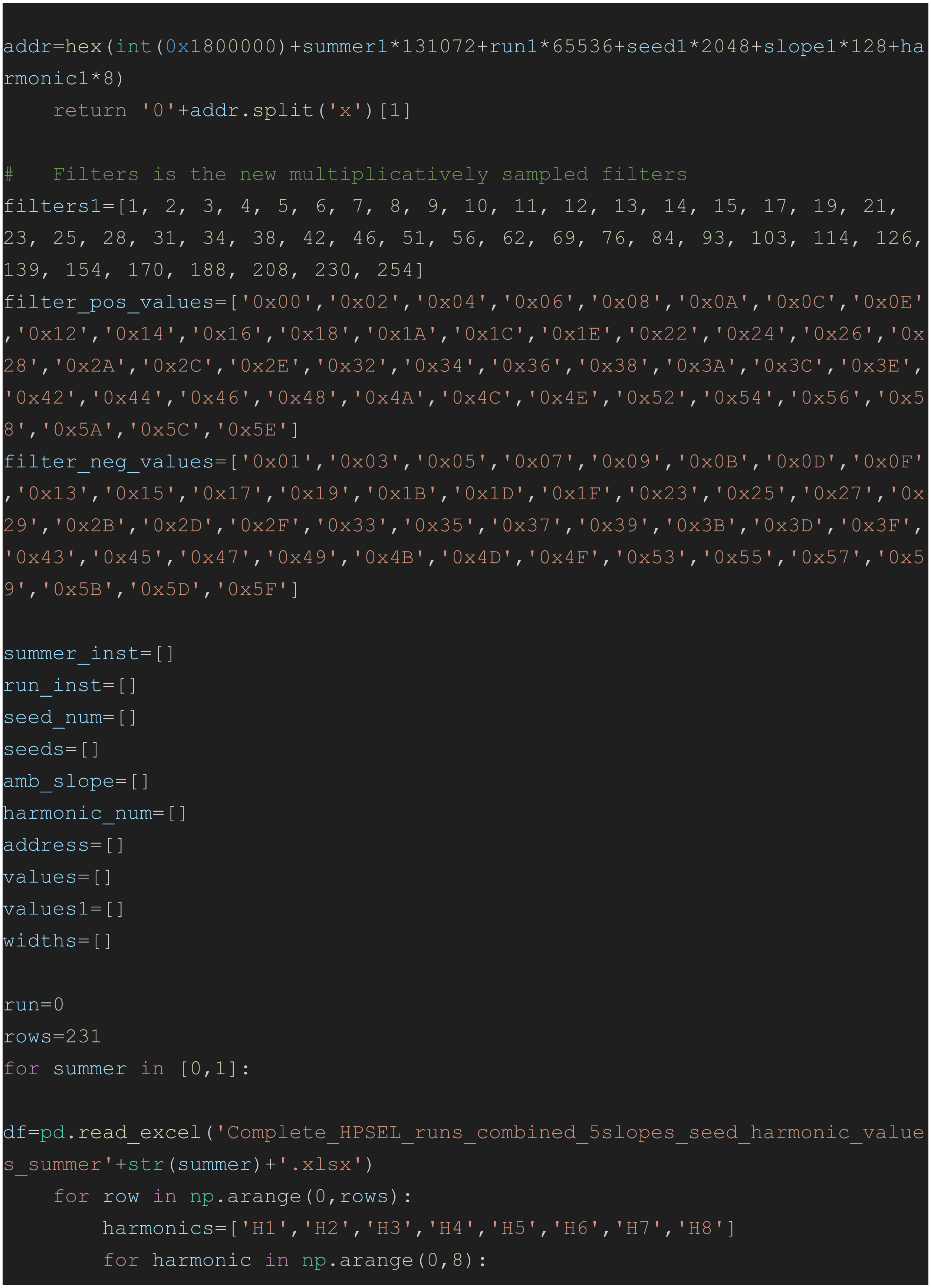
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd


def return_addr(summer1,run1,seed1,slope1,harmonic1):

addr=hex(int(0x1800000)+summer1*131072+run1*65536+seed1*2048+slope1*128+ha
rmonic1*8)
    return '0'+addr.split('x')[1]

# Read the run1 and run 2 spreadsheet for each summer instance. Create a
spreadsheet for each summer instance combining the two runs.

for summer in [0,1]:
    seed0=0
    seeds1=[]
    extra_padding=2
    S=[]
    S1=[]
    H1=[]
    H2=[]
    H3=[]
    H4=[]
    H5=[]
    H6=[]
    H7=[]
    H8=[]
    for run in [0,1]:

df=pd.read_excel('Complete_HPSEL_run'+str(run+1)+'_seed_slope_harmonic_val
ues_summer'+str(summer+1)+'.xlsx')
        seed1=run*21
        seed2=seed1+21
        for seed in np.arange(seed1,seed2):      ## Loop aims to read all
used rows in the given run
            for slope in range(0,11):
                row_idx=(seed-seed1)*11+slope
```



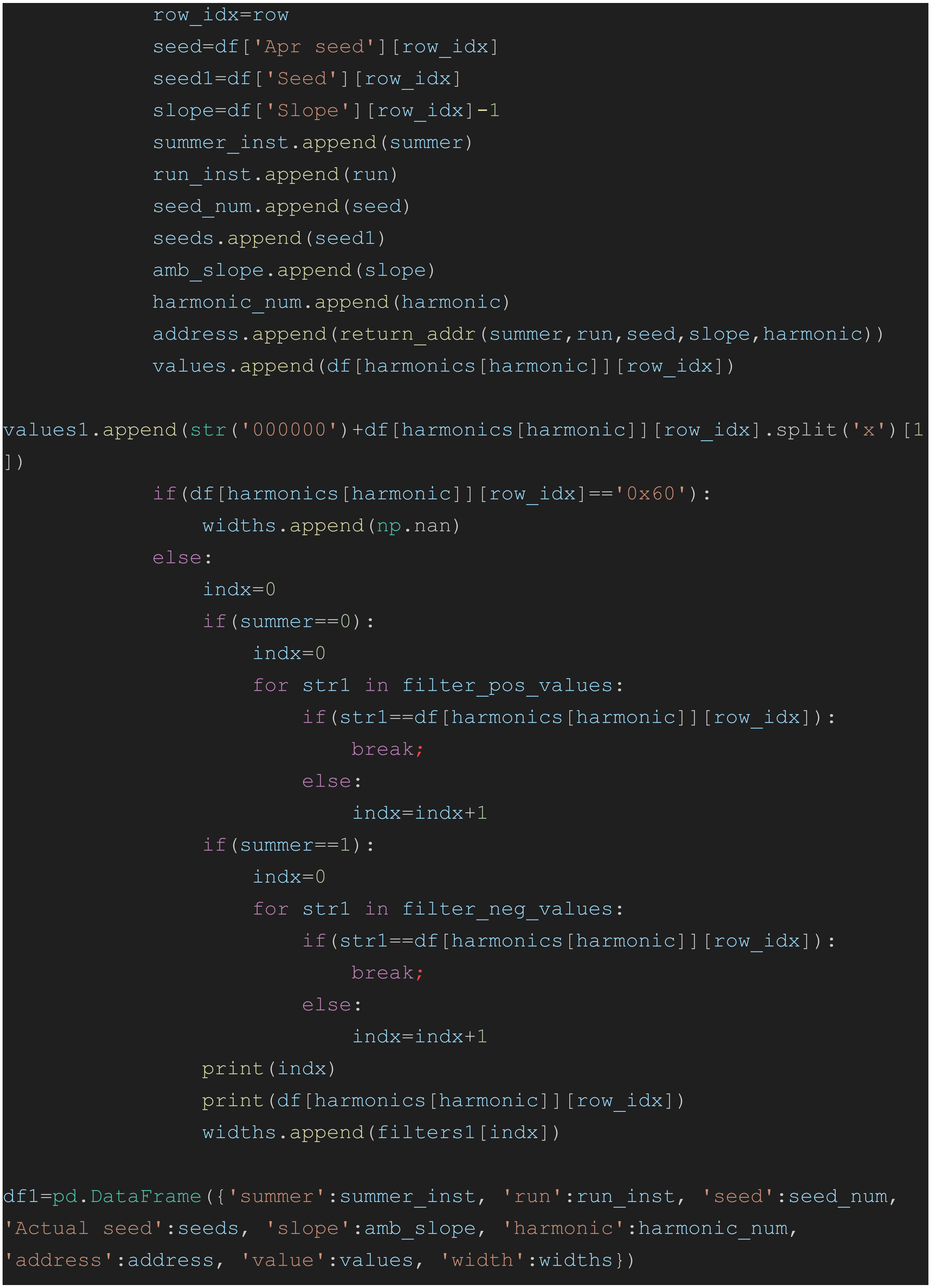
```
                if(df.loc[[row_idx]].to_numpy()[0][3] != '0x60'):
                    row_info=df.loc[[row_idx]].to_numpy()[0]
                    S.append(row_info[1])
                    Sl.append(row_info[2])
                    H1.append(row_info[3])
                    H2.append(row_info[4])
                    H3.append(row_info[5])
                    H4.append(row_info[6])
                    H5.append(row_info[7])
                    H6.append(row_info[8])
                    H7.append(row_info[9])
                    H8.append(row_info[10])

## Define empty lists to store the 5 slope configuration of the used rows
    rows=len(H1)
    S1=[]
    Sl1=[]
    H11=[]
    H21=[]
    H31=[]
    H41=[]
    H51=[]
    H61=[]
    H71=[]
    H81=[]
    i=0
    print(max(S))
    print(rows)
    for idx in range(rows):

        ##  Append the five consecutive used rows in the list
        S1.append(S[idx])
        Sl1.append(i+1)
        H11.append(H1[idx])
        H21.append(H2[idx])
        H31.append(H3[idx])
        H41.append(H4[idx])
        H51.append(H5[idx])
        H61.append(H6[idx])
        H71.append(H7[idx])
        H81.append(H8[idx])
        seeds1.append(seed0)
        i=i+1
        if(idx == rows-1):          ##  After the last used row, fill
remainign slopes with '0x60' value
            for j in range(11 - i):
                S1.append(S[idx])
                Sl1.append(i+j+1)
                H11.append('0x60')
                H21.append('0x60')
                H31.append('0x60')
                H41.append('0x60')
                H51.append('0x60')
                H61.append('0x60')
                H71.append('0x60')
                H81.append('0x60')
                seeds1.append(seed0)
        if(i == 5):               ## After 5 consecutive rows, append next
six rows with '0x60' value
            for j in range(6):
                S1.append(S[idx])
                Sl1.append(i+j+1)
                H11.append('0x60')
                H21.append('0x60')
                H31.append('0x60')
                H41.append('0x60')
                H51.append('0x60')
                H61.append('0x60')
                H71.append('0x60')
                H81.append('0x60')
                seeds1.append(seed0)
            i=0
            seed0=seed0+1

## 19 seeds were enough to capture all used summing instances. But to
complete the 21 seed config of a run, we do a padding for rest of 2 seeds
    for k in range(extra_padding):
        seed0=seed0+1
        i=0
        for j in range(11):
            S1.append(S[idx])
            Sl1.append(i+j+1)
            H11.append('0x60')
            H21.append('0x60')
            H31.append('0x60')
            H41.append('0x60')
            H51.append('0x60')
            H61.append('0x60')
            H71.append('0x60')
            H81.append('0x60')
            seeds1.append(seed0)

        i=0


## Create a new dataframe for the given summer instance and save it to a
file
    print(len(H1))
    print(len(seeds1))

    df1=pd.DataFrame({'Seed': S1, 'Slope':Sl1, 'H1':H11, 'H2':H21,
'H3':H31, 'H4':H41, 'H5':H51, 'H6':H61, 'H7':H71, 'H8':H81, 'Apr
seed':seeds1})

df1.to_excel('Complete_HPSEL_runs_combined_5slopes_seed_harmonic_values_su
mmer'+str(summer)+'.xlsx')
```

4. Complete_final_HPSEL_5slopes

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import os


def return_addr(summer1,run1,seed1,slope1,harmonic1):
```



```
addr=hex(int(0x1800000)+summer1*131072+run1*65536+seed1*2048+slope1*128+harmonic1*8)
    return '0'+addr.split('x')[1]

#   Filters is the new multiplicatively sampled filters
filters1=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 19, 21, 23, 25, 28, 31, 34, 38, 42, 46, 51, 56, 62, 69, 76, 84, 93, 103, 114, 126, 139, 154, 170, 188, 208, 230, 254]
filter_pos_values=['0x00','0x02','0x04','0x06','0x08','0x0A','0x0C','0x0E','0x12','0x14','0x16','0x18','0x1A','0x1C','0x1E','0x22','0x24','0x26','0x28','0x2A','0x2C','0x2E','0x32','0x34','0x36','0x38','0x3A','0x3C','0x3E','0x42','0x44','0x46','0x48','0x4A','0x4C','0x4E','0x52','0x54','0x56','0x58','0x5A','0x5C','0x5E']
filter_neg_values=['0x01','0x03','0x05','0x07','0x09','0x0B','0x0D','0x0F','0x13','0x15','0x17','0x19','0x1B','0x1D','0x1F','0x23','0x25','0x27','0x29','0x2B','0x2D','0x2F','0x33','0x35','0x37','0x39','0x3B','0x3D','0x3F','0x43','0x45','0x47','0x49','0x4B','0x4D','0x4F','0x53','0x55','0x57','0x59','0x5B','0x5D','0x5F']

summer_inst=[]
run_inst=[]
seed_num=[]
seeds=[]
amb_slope=[]
harmonic_num=[]
address=[]
values=[]
values1=[]
widths=[]

run=0
rows=231
for summer in [0,1]:

df=pd.read_excel('Complete_HPSEL_runs_combined_5slopes_seed_harmonic_values_summer'+str(summer)+'.xlsx')
    for row in np.arange(0,rows):
        harmonics=['H1','H2','H3','H4','H5','H6','H7','H8']
        for harmonic in np.arange(0,8):
```



```
            row_idx=row
            seed=df['Apr seed'][row_idx]
            seed1=df['Seed'][row_idx]
            slope=df['Slope'][row_idx]-1
            summer_inst.append(summer)
            run_inst.append(run)
            seed_num.append(seed)
            seeds.append(seed1)
            amb_slope.append(slope)
            harmonic_num.append(harmonic)
            address.append(return_addr(summer,run,seed,slope,harmonic))
            values.append(df[harmonics[harmonic]][row_idx])

values1.append(str('000000')+df[harmonics[harmonic]][row_idx].split('x')[1
])
            if(df[harmonics[harmonic]][row_idx]=='0x60'):
                widths.append(np.nan)
            else:
                indx=0
                if(summer==0):
                    indx=0
                    for str1 in filter_pos_values:
                        if(str1==df[harmonics[harmonic]][row_idx]):
                            break;
                        else:
                            indx=indx+1
                if(summer==1):
                    indx=0
                    for str1 in filter_neg_values:
                        if(str1==df[harmonics[harmonic]][row_idx]):
                            break;
                        else:
                            indx=indx+1
                print(indx)
                print(df[harmonics[harmonic]][row_idx])
                widths.append(filters1[indx])

df1=pd.DataFrame({'summer':summer_inst, 'run':run_inst, 'seed':seed_num,
'Actual seed':seeds, 'slope':amb_slope, 'harmonic':harmonic_num,
'address':address, 'value':values, 'width':widths})
```

```
df1.to_excel('Complete_HPSEL_final_addresses_values_single_run_5slopes.xls
x')
df2=pd.DataFrame({'address':address, 'values':values1})
df2.to_csv('Complete_HPSEL_only_addresses_values_single_run_5slopes.txt',s
ep='\t',index=False,header=False)

f1=open('Complete_HPSEL_only_addresses_values_single_run_5slopes.txt','r')
f2=open('FDAS_config.txt','r')
f3=open('Complete_HPSEL_only_addresses_values_single_run_5slopes_with_conf
ig.txt','a')
for line in f2:
    f3.write(line)

f3.write('\n')
for line in f1:
    f3.write(line)
f1.close()
f2.close()
f3.close()
```