Algorithm: denoise based on property of overtone

Let's call it "overtone-based denoise"

# Assumption:

# 1. noise do not overflow signal everywhere

# 2. (two times fundamental frequency) < N/2

# 3. Highest peak is at either fundamental frequency or the 1st overtone

**procedure** OBD(F) # input F is an array of DFT

A = magnitude of F;

N = length of A;

L = ;

Mark = array of zeroes with length N;

ave = sum(A) / N;

count = 5;

Find = 0;

# if after 5 search, no suitable peak is found: exit and report exception

while ((there is no 1 in Mark) && (count > 0)):

{

count--;

# circular-even -> only have to do left hand side

p = searchPeak(A, Mark, [1, L]);

# in reality, overtones are not precisely at integer multiples

range = interval[p-10, p+10];

# check this peak is 1st overtone or fundamental frequency

pf = searchPeak(A, Mark, 0.5\*range);

if A(pf) > sqrt(ave \* A(p)):

# that peak is the 1st overtone

range = interval[pf-10, pf+10];

if 3\*p < L: # see if we can search on

# those whose peak at 1st overtone usually has many overtones

# so we should be able to get its 2nd overtone

p2 = searchPeak(A, Mark, 3\*range);

if A(p2) > sqrt(A(p) \* ave): # still, use amplitude at p to check

# mark fundamental frequency & all overtones

for (i = 1; pf \* i < L; i++) :

Mark(searchPeak(A, Mark, i\*range));

Find = 1;

ff = pf; # fundamental frequency

else: # this is not funamental fequency

Mark(pf) = -1; # disqualify this peak

Mark(p) = -1;

else: # if we can't, then guess we find it.

Mark(pf) = 1;

Mark(p) = 1;

Find = 1;

ff = pf;

else: # the peak is at the fundamental frequency

# check if its 1st overtone exists

p1 = searchPeak(A, Mark, 2\*range);

if A(p1) >= sqrt(ave \* A(p)): # if so

if 3\*p < N/2:

p2 = searchPeak(A, 3\*range);

if A(p2) > sqrt(A(p1) \* ave):

for (i = 1; p \* i < L; i++) :

Mark(searchPeak(A, i\*range));

Find = 1;

ff = p;

else:

Mark(p) = -1;

else:

Mark(p) = 1;

Mark(p1) = 1;

Find = 1;

ff = p;

else: # this is not funamental fequency

Mark(p) = -1; # disqualify this peak

}

if (Find == 0):

output("No legal fundamental frequency found.");

return;

# Find all corresponding peaks on the other side

# In matlab, due to the way matlab index (1 to length(A)),

# circular-even is A[k] = A[length +2 - k]

for (i = 2; i <= L; i++)

if (Mark(i) == 1)

Mark(N + 2 - i) = 1;

keep = array of 0 with size N;

For all peaks P:

eps = findSpread(P, A, ff);

for (i = P - eps; i <= P + eps; i++):

keep(i) = 1;

For i = 1 to N:

if (keep(i) ! = 1)

F(i) = 0;

output("Procedure complete.")

return;

**end**

**procedure** searchPeak(A, Mark, [init, end])

# only for array with all elements nonnegative

# suppose no 2 elements have same value

max = 0; # value

maxI = -1; # index

for (i = init; i < end; i++):

if (Mark(i) != -1): # only if it is not disqualified

if (A(i) > max):

max = A(i);

maxI = i;

return maxI;

**end**

**procedure** findSpread(p, A, ff)

# each peak is at least ff's indices away from each other

ave = geometric mean of all points in A;

totalA = sum(A( : - ave);

# area under curve centered at peak with average deducted

eps = 1; # epsilon, the range of spread

AP = 0; # area so far

for (eps = 1; AP < 0.94 \* totalA; eps++):

# find spread by comparing the area ratio

AP = sum(A(p-eps : p+eps) - ave);

return eps;

**end**