Moorers reverb from Group 406

Reverb-20184317.ipynb

```
import numpy as np

import matplotlib.pyplot as plt

import scipy.io.wavfile as wave

import IPython.display as ipd

samplingFreq, dryMonsterSignal = wave.read('Data/monster.wav')

dryMonsterSignal = dryMonsterSignal/2**15 # Normalize

ipd.Audio(dryMonsterSignal, rate=samplingFreq)

nData = np.size(dryMonsterSignal)

timeVector = np.arange(nData)/samplingFreq # Seconds

plt.figure(figsize=(16,5))

plt.plot(timeVector,dryMonsterSignal,linewidth=2)

plt.xlim((timeVector[0],timeVector[-1]))

plt.xlabel('time [s]'), plt.ylabel('Amplitude');

freqVector = np.arange(nData)*samplingFreq/nData # Hz.

# Computing DFT using the FFT algorithm.

freqResponseDry = np.fft.fft(dryMonsterSignal)

ampSpectrumDry = np.abs(freqResponseDry)

plt.figure(figsize=(16,5))

plt.plot(freqVector,ampSpectrumDry,linewidth=2)

plt.xlim((0,2000))

plt.xlabel('freq. [Hz]'), plt.ylabel('Amplitude');

def lpFilter(inputSignal, filterParam):

    signalLength = np.size(inputSignal)

    outputSignal = np.zeros(signalLength)


    for n in np.arange(signalLength):
```

```
      outputSignal[n] = (1 - filterParam) * inputSignal[n-1] + filterParam * outputSignal[n-1]

   return outputSignal
def plainReverberator(inputSignal, delay, filterParam):

   signalLength = np.size(inputSignal)
   outputSignal = np.zeros(signalLength)

   for n in np.arange(signalLength):
      if n < delay:

         outputSignal[n] = inputSignal[n]

      else:

         outputSignal[n] = inputSignal[n] + filterParam*outputSignal[n-delay]

   outputSignal = lpFilter(outputSignal, filterParam)

   return outputSignal

monsterSignalWithPlainReverb = \
   plainReverberator(dryMonsterSignal, 1, 0.5) #Filter parameter/Gain can not be above 1 because of
instability.
ipd.Audio(monsterSignalWithPlainReverb, rate=samplingFreq)
nData = np.size(monsterSignalWithPlainReverb)
timeVector = np.arange(nData)/samplingFreq # Seconds.
plt.figure(figsize=(16,5))
plt.plot(timeVector,monsterSignalWithPlainReverb,linewidth=2)
plt.xlim((timeVector[0],timeVector[-1]))
```

```
plt.xlabel('time [s]'), plt.ylabel('Amplitude');
freqVector = np.arange(nData)*samplingFreq/nData # Hz.
# Computing DFT using the FFT algorithm.
freqResponseDry = np.fft.fft(monsterSignalWithPlainReverb)
ampSpectrumDry = np.abs(freqResponseDry)
plt.figure(figsize=(16,5))
plt.plot(freqVector,ampSpectrumDry,linewidth=2)
plt.xlim((0,2000))
plt.xlabel('freq. [Hz]'), plt.ylabel('Amplitude');
def allpassReverberator(inputSignal, delay, apParam):
    nData = np.size(inputSignal)
    outputSignal = np.zeros(nData)
    for n in np.arange(nData):
        if n < delay:
            outputSignal[n] = inputSignal[n]
        else:
            outputSignal[n] = apParam*inputSignal[n] + inputSignal[n-delay] - \
                apParam*outputSignal[n-delay]
    return outputSignal

monsterSignalWithAllPassReverb = \
    allpassReverberator(dryMonsterSignal, 1, 0.4)
ipd.Audio(monsterSignalWithAllPassReverb, rate=samplingFreq)
nData = np.size(monsterSignalWithAllPassReverb)
timeVector = np.arange(nData)/samplingFreq # Seconds.
plt.figure(figsize=(16,5))
plt.plot(timeVector,monsterSignalWithAllPassReverb,linewidth=2)
plt.xlim((timeVector[0],timeVector[-1]))
plt.xlabel('time [s]'), plt.ylabel('Amplitude');
freqVector = np.arange(nData)*samplingFreq/nData # Hz.
```

```
# Computing DFT using the FFT algorithm.
freqResponseDry = np.fft.fft(monsterSignalWithAllPassReverb)
ampSpectrumDry = np.abs(freqResponseDry)
plt.figure(figsize=(16,5))
plt.plot(freqVector,ampSpectrumDry,linewidth=2)
plt.xlim((0,2000))
plt.xlabel('freq. [Hz]'), plt.ylabel('Amplitude');
def plainGainFromReverbTime(reverbTime, plainDelay, samplingFreq):
    nDelays = np.size(plainDelay)
    plainGains = np.zeros(nDelays)
    for ii in np.arange(nDelays):
        plainGains[ii] = 10**(-3*plainDelays[ii]/(reverbTime*samplingFreq))
    return plainGains

def moorersReverb(inputSignal, mixingParams, plainDelays, plainGains, allpassDelays, apParams):
    nData = np.size(inputSignal)
    tmpSignal = np.zeros(nData)
    # Parallel plain reverberation
    nPlainReverberators = np.size(plainDelays)
    for ii in np.arange(nPlainReverberators):
        tmpSignal = tmpSignal + \
            mixingParams[ii]*plainReverberator(inputSignal, plainDelays[ii], plainGains[ii])
    # Serial all pass reverberation
    nAllpassReverberators = np.size(allpassDelays)
    for ii in np.arange(nAllpassReverberators):
        tmpSignal = allpassReverberator(tmpSignal, allpassDelays[ii], apParams[ii])
    return tmpSignal

# Sum of mixing parameters is one because that is the desired reverberation time.
mixPara1 = 0.20
```

```
mixPara2 = 0.15

mixPara3 = 0.20

mixPara4 = 0.10

mixPara5 = 0.20

mixPara6 = 0.15


# Large delays through prime numbers, so that their non-zero output rarely overlaps.

plainD1 = 1487

plainD2 = 1319

plainD3 = 1607

plainD4 = 1583

plainD5 = 1657

plainD6 = 1789


# Small delays through prime numbers, to increase echo density but not increase reverberation time.

allPassD1 = 227

allPassD2 = 223


# Not too close to 1, because we want the reverberation to die out.

allPassP1 = 0.70

allPassP2 = 0.80


mixingParams = np.array([mixPara1, mixPara2, mixPara3, mixPara4, mixPara5, mixPara6])

plainDelays = np.array([plainD1, plainD2, plainD3, plainD4, plainD5, plainD6])

allpassDelays = np.array([allPassD1, allPassD2])

apParams = np.array([allPassP1, allPassP2])

reverbTime = 0.01 # Seconds

plainGains = plainGainFromReverbTime(reverbTime, plainDelays, samplingFreq)

# Computing the impulse response of a room.

irLength = np.int(np.floor(reverbTime*samplingFreq))
```

```
impulse = np.r_[np.array([1]),np.zeros(irLength-1)]

impulseResponse = monsterSignalWithAllPassReverb = \

   moorersReverb(impulse, mixingParams, plainDelays, plainGains, allpassDelays, apParams)


monsterSignalWithMoorerReverb = \

   moorersReverb(dryMonsterSignal, mixingParams, plainDelays, plainGains, allpassDelays, apParams)

ipd.Audio(monsterSignalWithMoorerReverb, rate=samplingFreq)

nData = np.size(monsterSignalWithMoorerReverb)

timeVector = np.arange(nData)/samplingFreq # Seconds.

plt.figure(figsize=(16,5))

plt.plot(timeVector,monsterSignalWithMoorerReverb,linewidth=2)

plt.xlim((timeVector[0],timeVector[-1]))

plt.xlabel('time [s]'), plt.ylabel('Amplitude');

freqVector = np.arange(nData)*samplingFreq/nData # Hz.

# Computing DFT using the FFT algorithm.

freqResponseDry = np.fft.fft(monsterSignalWithMoorerReverb)

ampSpectrumDry = np.abs(freqResponseDry)

plt.figure(figsize=(16,5))

plt.plot(freqVector,ampSpectrumDry,linewidth=2)

plt.xlim((0,2000))

plt.xlabel('freq. [Hz]'), plt.ylabel('Amplitude');
```