Lossy Compression Project: Issued Dec. 2, Due Dec. 17 at 4:30 p.m.

Overview

Sources such as audio and video are typically compressed using lossy, rather than lossless, compression. The reason is that a small amount of distortion is usually tolerable, and allowing a small amount of distorton yields significantly more compression. The goal of this project is to design an encoder and decoder for lossy compression of audio.

We will focus on compressing one particular audio file, a recording of Neil Armstrong's famous line "That's one small step for [a] man, one giant leap for mankind." A `wav`-format (i.e., uncompressed) recording is about 652,000 bytes long. The goal is to compress it as much as possible.

Your assignment is to write a Python script that, after reading from a data file that you create, creates a properly-formatted `.wav` file for which the time-average mean square error is no more than a threshold specified below. We view the data file as the compressed version of the audio and the Python script as the decoder.

Rules

You are asked to submit exactly three files: a data file, a decompression program, and a report. The data file must be named `compressed` and the decompression program must be named `decompress.py`.

We place no constraints whatsoever on the `compressed` file. You may place whatever data you like in this file in whatever format you like. The file `decompress.py`, on the other hand, must be a Python script that runs under the Python 3.12.4 installation on the ECE School's Linux cluster (`ecelinux.ece`), which is the version of Python that is run if you type `module load anaconda3` and then `python`.

Your Python script may only import modules from the following list

```
struct
wave
numpy
scipy
decimal
```

The `struct` module is useful for basic I/O. The `wave` module is useful for reading and writing `.wav` files. The `numpy` and `scipy` provide useful numerical routines, including the DFT and DCT. The `decimal` module is useful for arithmetic coding.

The `decompress.py` script must create a `.wav` file named `out.wav` containing exactly the same number of samples as the master file `step.wav`, which is posted on the course Canvas site. The normalized mean square error (MSE) between the `out.wav` and `step.wav` must be no more than $4 \cdot 10^{-5}$. There is a Python script, `computemse.py`, that will compute the MSE automatically for you.

Your decompression program may not use more than 20 minutes of CPU time in user mode, nor may it call external executables or import any modules other than those listed above. You can determine how many CPU-seconds your program spends in user mode using the command

```
% /usr/bin/time --format %U python decompress.py
```

We define the length of your compressed version of the audio file to be the *sum* of the file sizes of `compressed` and the size of your decompressor after it has been compiled to Python bytecode. This compilation can be accomplished in the Python interpreter via the commands

```
>>> import py_compile
>>> py_compile.compile('decompress.py')
```

This will create a file

```
__pycache__/decompress.pyc
```

consisting of the compiled bytecode. This *Kolmogorov complexity* approach (cf. Chapter 14 in Cover and Thomas) is necessary because one can embed information about the audio recording in the Python script itself. Likewise, you are not permitted to use multiple data files, or to call your data file anything other than `compressed`, because the names of the data files or the way in which data is divided among them could be used to store some of the audio.

Submission

The project is due on Tuesday, Dec 17th at 4:30 p.m. You are asked to electronically submit your `compressed` and `decompress.py` files and your report via the Canvas site. The report, which should be in PDF format, should describe your approach in sufficient detail to enable the reader to write a functionally equivalent version of your algorithm without looking at your code. Please also describe any novel features of your solution.

No late submissions are permitted for this project.

Collaboration

You may submit an individual solution to this project, or a joint project with one other student in the course. Either way, you are encouraged to discuss this project with others, including current and former students in the course. You are also encouraged to consult web resources and the published literature. You can find research papers on lossy compression through the Engineering Library's website:

`http://engineering.library.cornell.edu/`

In your written submission, however, *you must include a separate "acknowledgment" section in which you list the sources of all ideas that you used in your final solution that are not your own.*

Grading

40% of the overall score will be determined by the compression level of the solution. Solutions that do not meet the distortion constraint or that require more than 20 minutes of CPU time will receive none of this credit. 35% will be determined by the quality of the technical approach. 20% will be determined by the quality and completeness of the written report and the commenting of the source file. 5% will be given simply for acknowledging others' ideas (if the ideas are entirely your own, simply state this to receive the 5%).

Tips

1. In addition to the master audio file, `step.wav`, the course Canvas site contains the following files:

   (a) `decompress.py`: a sample decoder that meets the distortion contraint

   (b) `compressed`: the data file used by the above decoder

   (c) `compress.py`: the encoder that produced the `compressed` file

   (d) `computemse.py`: A Python script that computes the MSE between two `.wav` files, assumed to be named `step.wav` and `out.wav`

   The sample solution compresses the audio by computing the DFT of the entire recording, retaining only the lowest frequency components, and encoding them using a 16-bit uniform quantizer. It uses the SciPy module's efficient implementation of the DFT. A naive implementation of the DFT seems to violate the CPU-time limit. Documentation for the SciPy module is available at

`https://docs.scipy.org/doc/scipy/reference/`

2. Since you are not being asked to submit a program that creates `compressed`, your encoding does not need to be done via a Python script. For this you are welcome to use UNIX tools or programs written in other languages such as MATLAB. You should describe how this file was created in your report, however. Also note that the 20-minute limit is on the *decoding* process. Encoding can take as long as you like.

3. The sample solution can be improved in many ways. As discussed in class, it is advantageous to perform the DFT on smaller blocks to create a sequence of samples at each frequency. Those sequences of samples can then be vector quantized, using the waterfilling formula to decide how much rate to use for each component. Even better, one could use the DCT instead of the DFT. Note that the SciPy module also provides a fast implementation of the DCT.

4. As discussed in class, scalar quantization of transform coefficients followed by lossless compression (i.e., "entropy coding") performs almost as well as vector quantization. If you decide to do lossless compression of transform coefficients, feel free to reuse code that you developed for the lossless compression project.

5. Bear in mind that standard audio compression formats such as mp3 and AAC are optimized for the human ear, not for mean square error. They also include many features that are not relevant for this project. Thus, while you are allowed to use code from mp3 and AAC decoders in your solution (so long as you acknowledge it and explain in sufficient detail how it works), you will probably obtain better results by using the sample solution on the Canvas site as your starting point.

6. For the purposes of this project, it is disadvantageous to include error checking code in your solution. We will not be applying your compression algorithm to arbitrary `.wav` files and including error checking in your code, while usually a good programming practice, while increase the size of your decompressor.

Good luck!