# Compressing Audio with the Discrete Cosine Transform

## Introduction

This application demonstrates how you can compress a signal by discarding low-energy parts of its discrete cosine transform. Specifically, we only retain those coefficients that cumulatively sum to a large part of the signal energy.

Here, the signal is an audio file, where only 13% of the DCT coefficients are needed to represent 97% of the signal energy. After compression, the resulting audio is hissy but still legible.

This loudspeaker is needed to play the audio

```
> restart:
  with(SignalProcessing):
  with(AudioTools):
  with(ColorTools):
```

```
> common_plot_opts :=
   axes              = boxed
  ,axesfont          = [Calibri]
  ,size              = [800, 400]
  ,legendstyle       = [font = [Calibri]]
  ,labeldirections = [horizontal, vertical]
  ,labelfont         = [Calibri]
  ,titlefont         = [Calibri, 16]
  ,background        = Color("RGB", [218/255, 223/255, 225/255])
  ,axis              = [gridlines = [5, color = Color("RGB", [1, 1,
  1])]]:
```

## Import and Play Audio

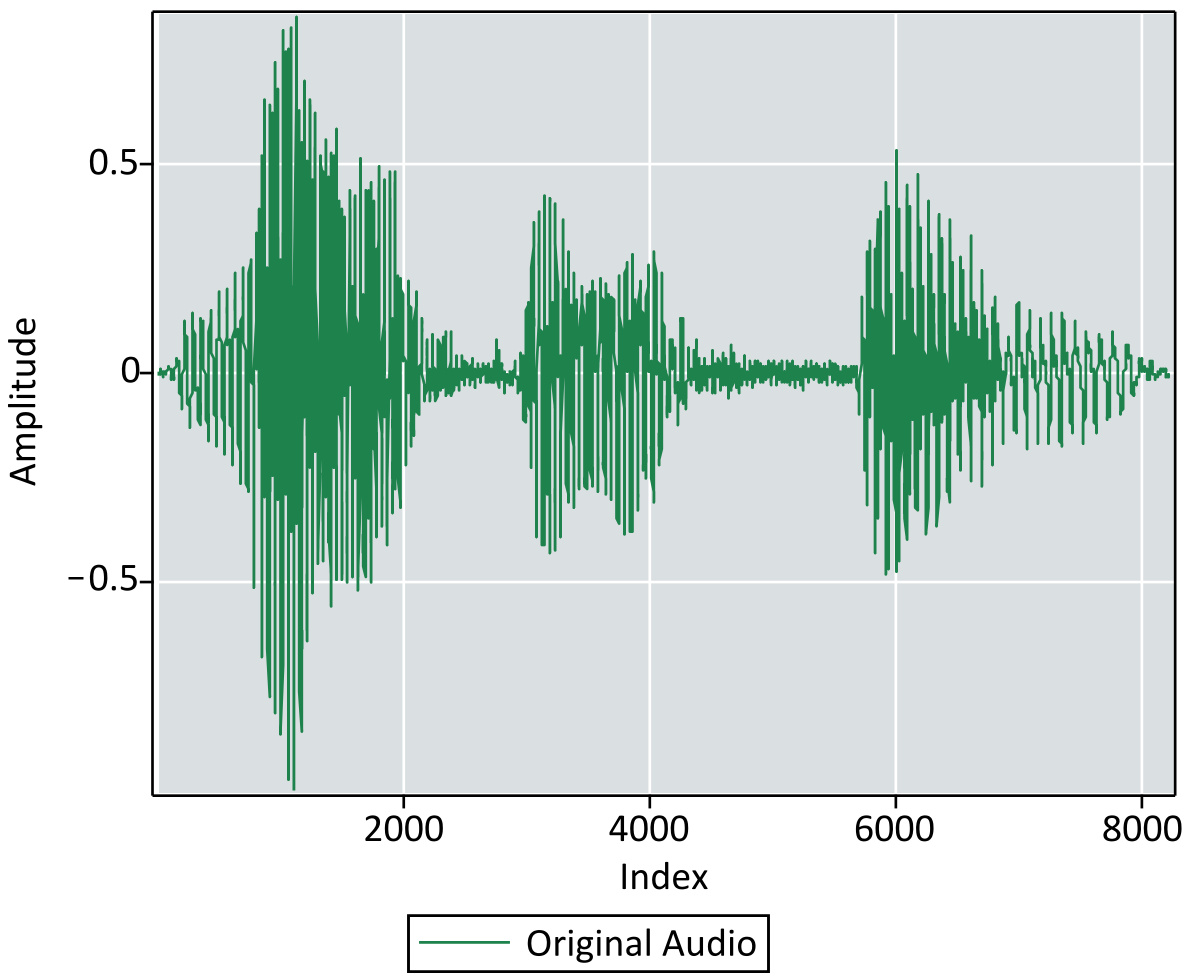
```
> aud := Read(FileTools:-JoinPath([kernelopts(datadir), "audio",
  "maplesim.wav"]));
  Fs  := attributes(aud)[1];
```

```
Play(aud)
```

$$aud := \begin{bmatrix} \text{"Sample Rate"} & 11025 \\ \text{"Bit Depth"} & 16 \\ \text{"Channels"} & 1 \\ \text{"Points/Channel"} & 8227 \\ \text{"Duration"} & 0.75\,s \end{bmatrix}$$

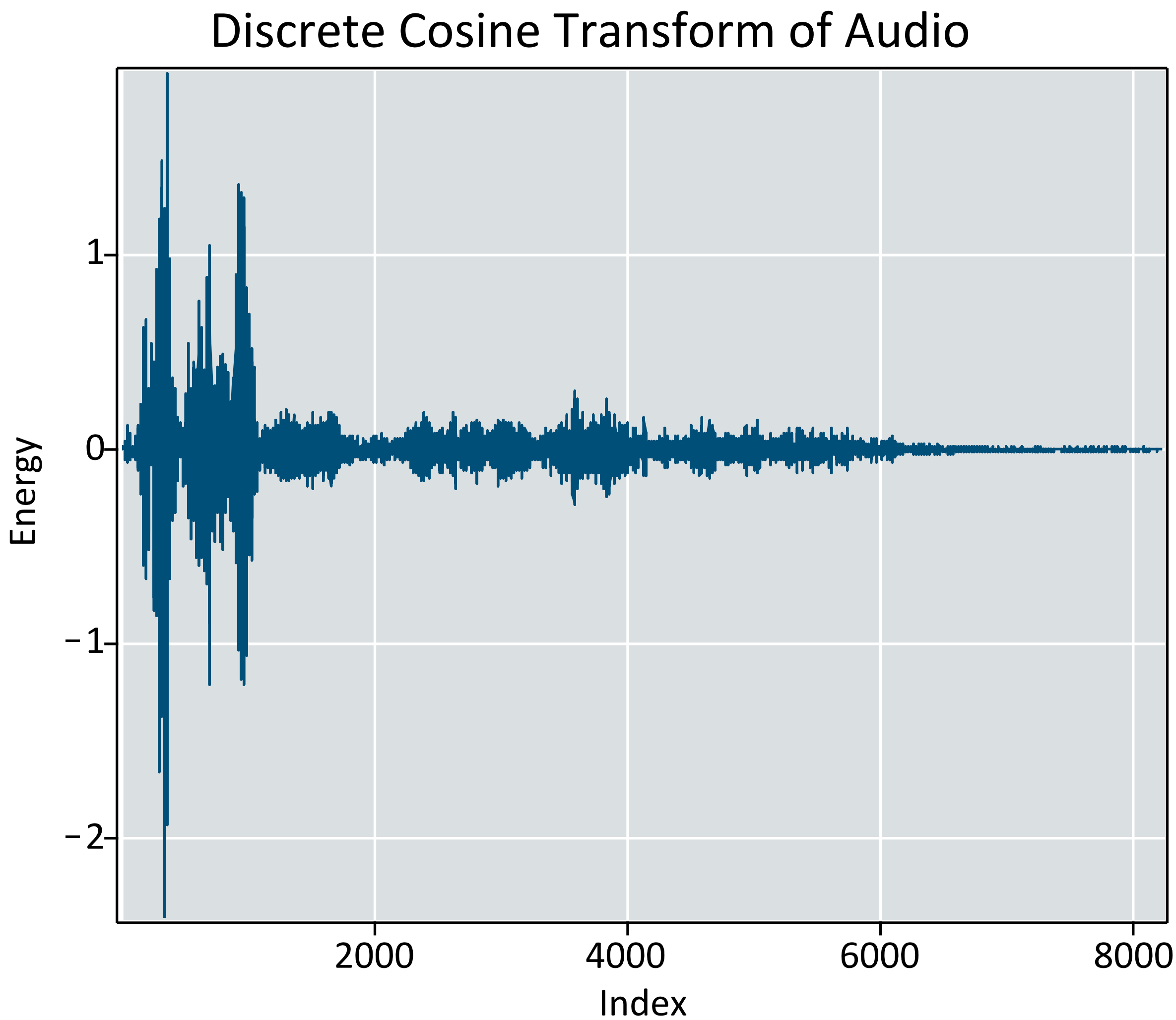
$$Fs := 11025 \qquad (2.1)$$

```
> p1 := dataplot(aud, style = line, thickness = 0, color = Color
  ("RGB",[30/255, 130/255, 76/255]), legend = "Original Audio",
  labels = ["Index", "Amplitude"], common_plot_opts)
```




## Calculate the Direct Cosine Transform

```
> aud_dct := DCT(aud):
  dataplot(aud_dct, style = line, thickness = 0, color = Color
  ("RGB",[0/255, 79/255, 121/255]), labels = ["Index", "Energy"],
  title = "Discrete Cosine Transform of Audio", common_plot_opts)
```




## Calculate the number of DCT coefficients needed to model 97% of the energy

Sort the DCT coefficients into descending order (i.e. the coefficients that represent the most signal energy first)

```
> ind := sort(abs(aud_dct), `>`, output = permutation):
```

Calculate how many of the sorted coefficients are needed to retain 97% of the energy

```
> num_coeffs := 1:

  do num_coeffs++ until Norm(aud_dct[ind[1..num_coeffs]], 2) /
  Norm(aud_dct, 2) > 0.97:

  num_coeffs
```

$$1074 \quad (4.1)$$

13% of the DCT coefficients are needed to retain 97% of the signal energy

```
> evalf(num_coeffs / numelems(aud_dct))
```

$$0.1305457639 \quad (4.2)$$

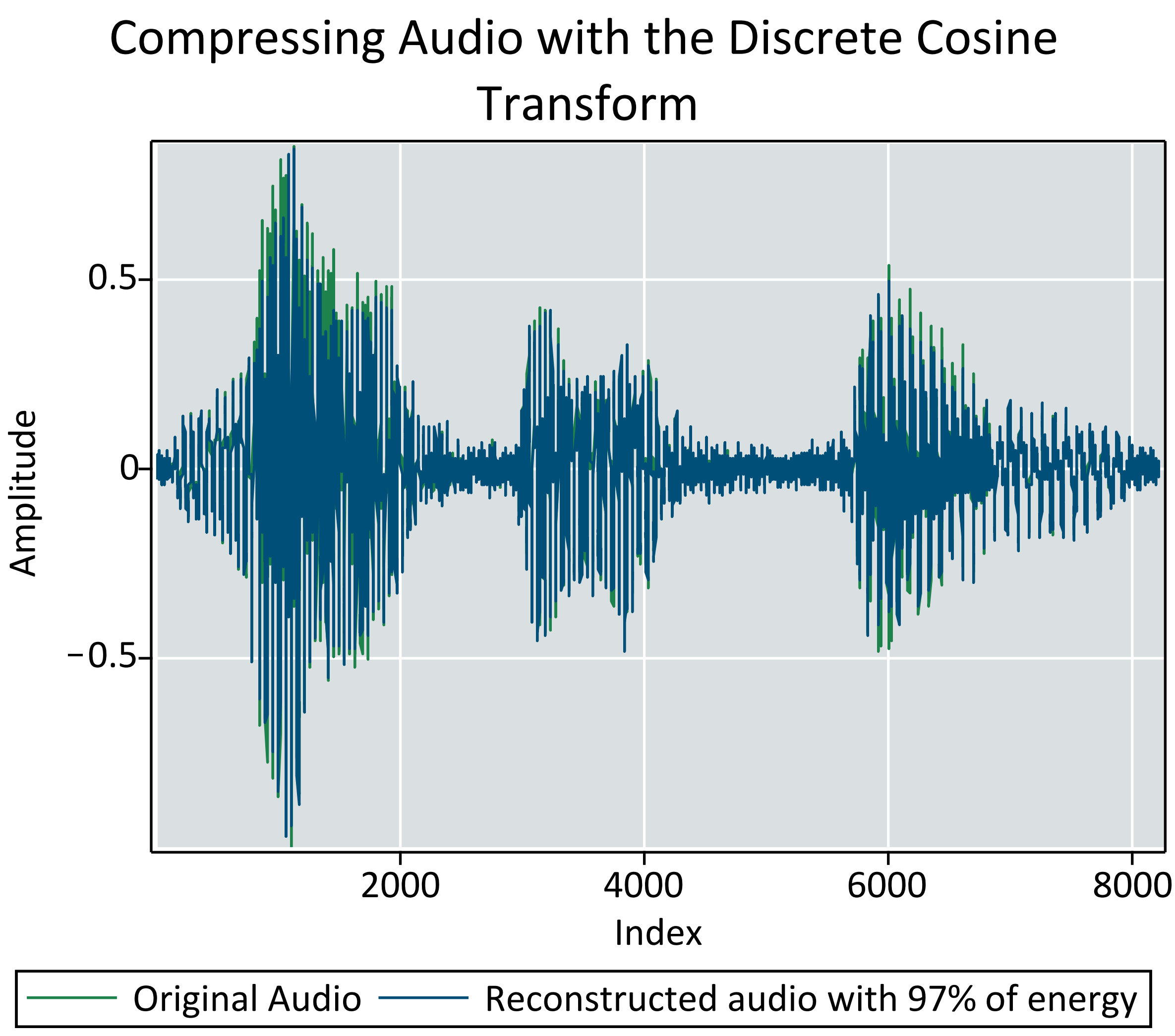
Set the remaining coefficients to 0

```
> aud_dct[ind[num_coeffs + 1..]] := 0:
```

## Reconstruct and Play the Compressed Audio

```
> aud_recon := InverseDCT(aud_dct):

> p2 := dataplot(aud_recon, style = line, thickness = 0, color =
  Color("RGB",[0/255, 79/255, 121/255]), legend = "Reconstructed
  audio with 97% of energy", title = "Compressing Audio with the
  Discrete Cosine Transform"):
  plots:-display(p1, p2, common_plot_opts)
```




```
> Play(Create(aud_recon, rate = Fs))
```

The reconstructed audio is hissy, but is still legible