



دانشکده مهندسی کامپیوتر

انتقال داده

پاییز ۱۴۰۰

پروژه دوم

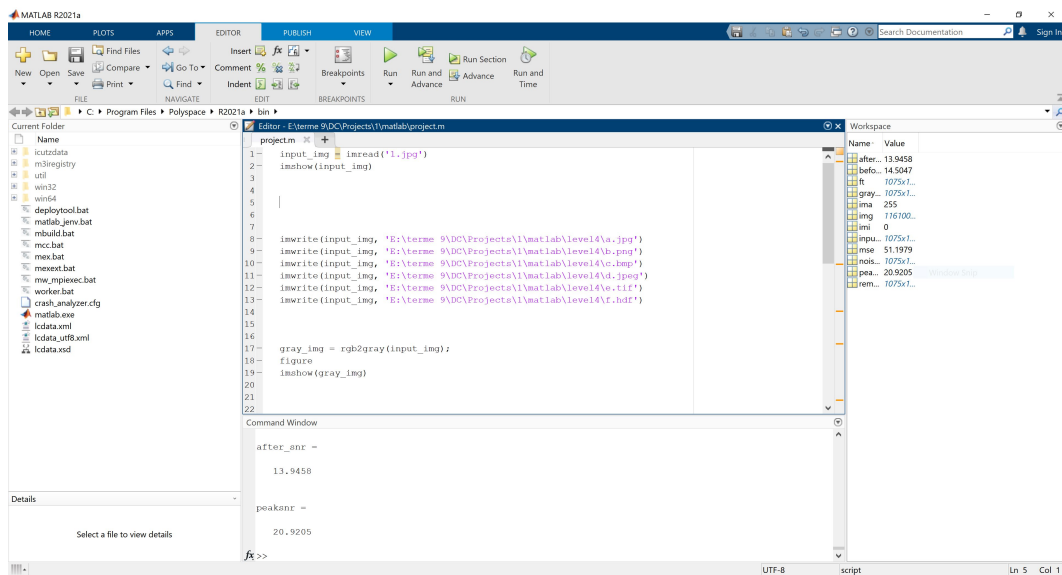
Voice processing

استاد درس دکتر ابوالفضل دیانت

تدوین ۹۶۴۶۲۱۰۴ - محمد یارمقدم

گام اول

در ابتدا به نصب نرم افزار Matlab پرداختم. برای این کار نسخه R2021a را نصب کردم و از طریق کرک موجود در آن به انجام مراحل فعالسازی پرداختم.



شکل ۱: عکس ورودی

گام دوم

در این مرحله با استفاده از دستور `audioread` ویس موردنظر را از کامپیوتر لوکال خوانده و وارد نرم افزار matlab کردم. این ویس با فرمت `.wav` بود و وارد شد.

```
clc  
clear  
%STEP 1  
[data, fps] = audioread('temp.wav');
```

شکل ۲: دستور در محیط Matlab

گام سوم

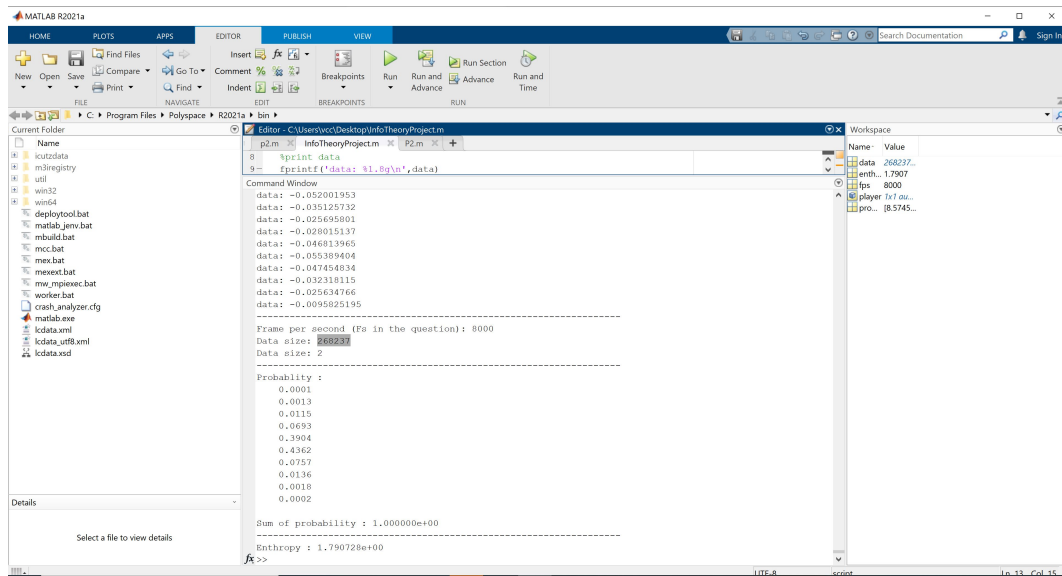
در این مرحله ابتدا داده های ورودی را پرینت می کنیم تا سوال اول این گام را پاسخ دهیم. الفبای منبع هشت بیتی هستند و ۷ رقم اعشار دقت دارند و تعدادی برابر با ۲۶۸۲۳۷ دارند. نوع داده های ورودی نیز از نوع Double است. برای یافتن تعداد داده های ورودی ابتدا آنها را در یک آرایه ذخیره و سپس سائز آرایه را پرینت کردم. سپس مقدار fps را پرینت کردم. این مقدار را می توانیم از آرگومان های داده شده توسط تابع audioread بدست آوریم. این مقدار در تابع فابل ورودی ۸۰۰۰ است که اگر تعداد کل ورودی را بر آن تقسیم کنیم زمان فایل را خواهیم داشت. که این مقدار برابر ۸.۳۲ است. اگر بخواهیم صوت ورودی در زمان پخش مشکلی نداشته باشد، باید طبق قضیه نایکوئیست نرخ نمونه برداری بیشتر از دو برابر بزرگترین مقدار فرکانس باشد. از آنجا که ذهن انسان تنها توانایی پردازش ۱۲ تا ۱۳ فریم در ثانیه را دارد فیلم تولید شده با ۲۰ تا ۳۰ فریم در ثانیه به صورت پیوسته تصور می شود و قطع شدن آن از چشم ما پنهان می ماند.

```
%STEP 3
%print data
fprintf('data: %1.8g\n',data)
fprintf("-----\n")
% get frame per second
fprintf("Frame per second (Fs in the question): %d\n", fps)
%size of data
fprintf("Data size: %d\n", size(data))
```

شکل ۳: دستورات استفاده شده در گام سوم

داده ورودی ما دو کانال دارد. برای آنکه در گام پنج تابع huffmandict فقط داده یک بعدی را به عنوان ورودی دریافت می کند پس از راه زیر آن را تک بعدی می کنیم. $data_1 = input_data(:, 1)$

خروجی محیط متلب



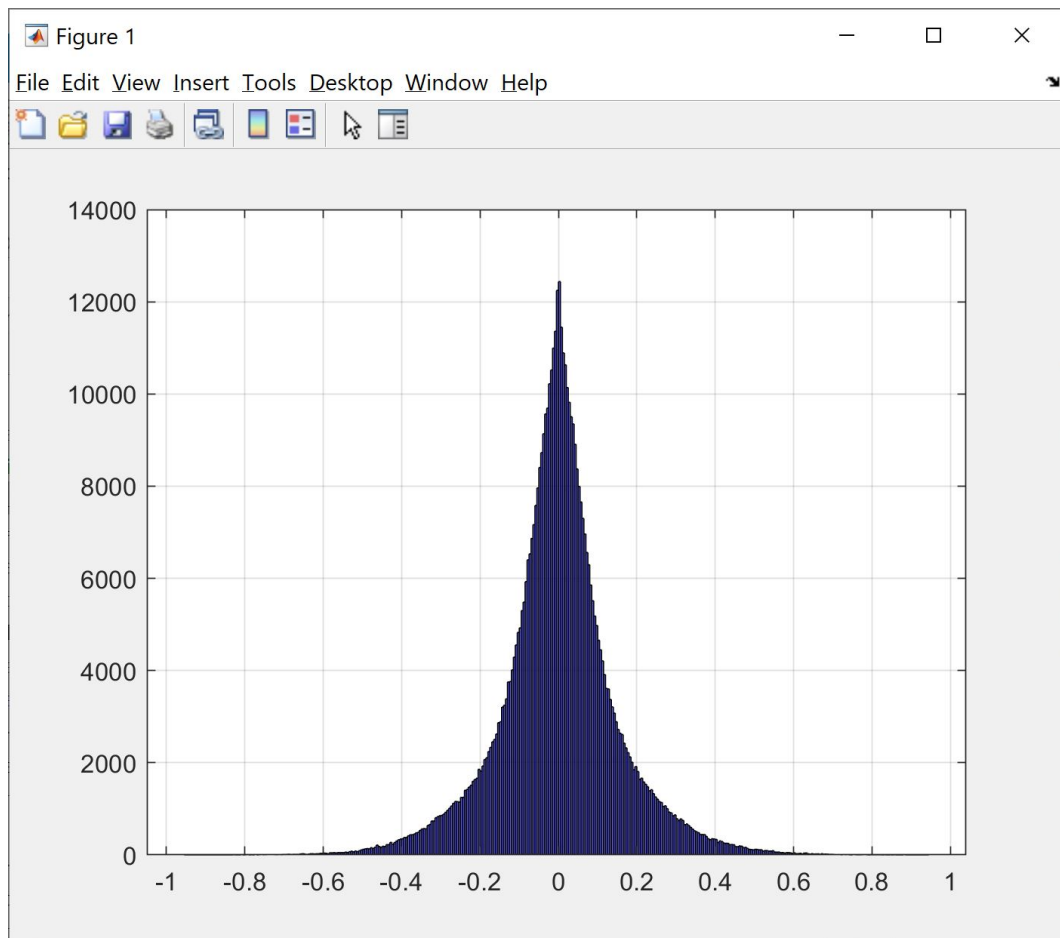
```
p2.m | InfoTheoryProject.m | P2.m | +
8 %print data
9 = fprintf('data: %1.8g\n',data)

Command Window
data: -0.052001953
data: -0.035125732
data: -0.025695801
data: -0.028015137
data: -0.046813965
data: -0.055389404
data: -0.047454834
data: -0.032318115
data: -0.025634766
data: -0.0095825195
-----
Frame per second (Fs in the question): 8000
Data size: 86633
Data size: 2
-----
Probability :
0.0001
0.0013
0.0115
0.0693
0.3904
0.4362
0.0757
0.0136
0.0019
0.0002
-----
Sum of probability : 1.000000e+00
-----
Entropy : 1.790728e+00
fx>>
```

شکل ۴: خروجی سه گام ابتدایی

گام چهارم

در این مرحله هیستوگرام فایل صوت ورودی را با فرض مستقل بودن سمپل های منبع از یکدیگر رسم کردم. این کار را با استفاده از دستور histogram انجام دادم. خروجی این مرحله را در قسمت زیر مشاهده می کنید.



شکل ۵: Histogram

پس از آن مقدار آنتروپی را محاسبه کردم. برای اینکار ابتدا احتمای ورودی در کل را محاسبه و سپس طبق رابطه آنتروپی آنرا محاسبه کردم. این رابطه یک رابطه لگاریتمی است. مقدار آنتروپی برابر $1.790728e+00$ شد.

محاسبه حداکثر مقدار فشرده سازی

برای این کار مقدار آنتروپی را به مدت زمان فایل که در واقع حاصل تقسیم تعداد کل ورودی به fps است، بدست می آوریم. پس داریم:

$$1.790728 * 268237 / 8000 = 600.424383$$

با توجه به قضیه اول شانون می توان این فایل صوتی را تا مقدار بالا می توان فشرده کرد. نتیجه به دست آمده منطقی نمی باشد چرا که از طرفی آنتروپی به دست آمده از فایل نمونه برداری شده به دست آمده و لزوما دقت لازم را دارا نیست و از طرفی دیگر چون این فایل حاصل نمونه برداری است مقادیر سطوح انرژی می توانند در بازه های کوتاه زمانی بسیار نزدیک به هم باشند و این تفاوت نادیده گرفته می شود از طرفی احتمال ظاهر شدن سمبل های منبع در کنار یکدیگر نیز مستقل نبوده و فرض مستقل بودن آن ها مناسب نیست و به هم وابستگی دارند از این رو این مرز دقیق و مناسب نیست.

```
%STEP 4
histogram(input_data, 'FaceColor', 'blue');
grid on;
%Getting first channel of input data
%Input wav file has two channels and we need 1D dimensional array
data_1 = input_data(:,1);
probability = hist(data_1)/sum(hist(data_1));
size_of_probs = size(probability)
fprintf("Probablity : \n")
disp(probability)
%Get sum of all probs
fprintf("Total probability: %d\n", sum(probability))
fprintf("-----End of Printing Probs-----\n")
entropy = -sum(probability .* log2(probability));
fprintf("Entropy: %d\n", entropy)
```

شکل ۶: کد بخش محاسبه آنتروپی

گام پنجم

در این مرحله ما کدگذاری هافمن را پیاده سازی کردیم. برای این امر از تابع `huffmandict` استفاده کردم. همانطور که در گام دوم اشاره شد داده را برای دادن به این تابع تک بعدی کردم.

```
%STEP 5|
fprintf("-----End of Entrophy calculations-----\n")
disp(hist(data_1));
%display histogram of first channel of data
dict = huffmandict(hist(data_1),probability);
sig = randsrc(268237,1,[hist(data_1);probability]);
%Calculate size of compressed file
comp = huffmanenco(sig,dict);
% convert to rows
fprintf("huffman compressed encode: %d\n", comp)
comp = comp';
audiowrite('compressed_temp.wav', comp, fps);
```

شکل ۷: کد بخش هافمن

کل حجم صوت ورودی ۱۰۲۰ کیلوبایت است و مرز شانون ۶۰۱ کیلوبایت است. پس از کدگذاری هافمن به ۹۴۹ کیلوبایت نیاز داریم که به مرز شانون نزدیکتر است. در قسمت آخر نیز برای محاسبه مقدار زمان انتقال طبق روابط زیر عمل می کنیم:

$$64 \text{ Kbit/s} / 8 = 8 \text{ Kbyte/s} \rightarrow 949 / 8 = 118.625 \text{ s}$$

کیفیت فایل خروجی اما جالب نیست و احتمالاً با حذف داده کاهش حجم اتفاق افتاده است پس می تواند روش خوبی نباشد. یک مشکل هم این است که تابع `huffman` روی متغیر `int` کار می کند ولی مقادیر صوت ورودی بین ۱ و -۱ است و قابلیت `cast` نخواهد داشت.

جمع بندی

در این پروژه با مفاهیم آنتروپی و فریم بر ثانیه و حجم صوت و ابعاد آن آشنا شدیم و سپس روش هافمن را روی آن پیاده سازی کردیم و الگوریتم کدگذاری آن را تمرین کردیم. گزارش پروژه هم با لتکس نوشته و آماده سازی شد.
لینک پروژه overleaf:

<https://www.overleaf.com/project/61c5c6b8599608df2bf28405>