دانشکده کامپیوتر

پروژه انتقال داده

دکتر اکبری

شایان موسوی نیا ۹۷۵۲۲۲۳۸

نیر ۱۴۰۱

```
%Part2
clc
clear
[y, Fs] = audioread ('Recording.wav');
player = audioplayer (y,Fs);
play(player);

```

شکل ۱: کدهای مربوط به گام دوم:

گام دوم

```
 8      %Part3
 9 -    disp(length(y))
10 -    disp(Fs)
11
```

شکل ۲: کدهای مربوط به گام سوم

```
263168

 48000
```

شکل ۳: طول و سرعت

گام سوم

پرسش اول:

فایل صوتی آرایه ای با ۲۱۳۶۶۸ نمونه از اعداد اعشاری با دقت ۱۵ رقم اعشار است.

پرسش دوم:

سرعت منبع اطلاعات (Fs) ۴۸۰۰۰ سمبل بر ثانیه است. با تقسیم ۲۱۳۶۶۸ بر ۴۸۰۰۰ زمان تقریبی فایل صوتی یعنی ۵.۴ ثانیه بدست می آید.

پرسش سوم:

چشم انسان با هماهنگی بخش بینایی مغز خواهد توانست تفاوت میان فریم‌های بیش از ۶۰ را تشخیص دهد اما این میزان تفاوت به حدی برای چشم انسان ناچیز به نظر می‌رسد که عملا نرخ فریم‌های بالاتر از ۶۰ عدد در هر ثانیه غیر قابل فهم به نظر می‌رسند. طبق تحقیقات دانشمندان و ثبت نوار Electroencephalogram که به اختصار EEG نامیده می‌شود، مغز انسان تنها تا ۱۳ فریم بر ثانیه را درک می‌کند و مقادیر بالاتر از آن غیر قابل درک خواهند بود. درست است که انسان می‌تواند حتی تفاوت فریم ریت بیش از ۶۰ را نیز تشخیص دهد، اما این به معنای آن نیست که ما واقعا همه‌ی فریم‌ها را درک می‌کنیم. به بیانی دیگر حداکثر تا مرز ۱۳ عکس بر ثانیه ما با دیدن تصاویری که مرتبط و به دنبال هم هستند، می‌توانیم مجزا بودن این تصاویر را از یکدیگر تشخیص دهیم و مغز ما فریب نمی‌خورد اما پس از حداکثر ۱۳ عکس بر ثانیه سیستم عصبی ما تصور می‌کند که تصاویر ذکر شده به صورت پیوسته به هم هستند. همین موضوع راجع به گوش انسان نیز صدق می کند و پرده گوش انسان آن قدر سریع حرکت نمی کند و فرکانس ها را سریع تشخیص نمی دهد که متوجه قطعی صدا بشویم.همچنین مدارانی در player ها وجود دارد که سیگنال گسسته را به آنالوگ تبدیل می کند.

```
%Part4
hist = histogram(y, 'Normalization' , 'Probability');
hisgrm = hist.Values;
hisgrm= hisgrm./length(y);
X = hisgrm;
l=length(X);
entro= 0;
for i=1:l
     if X(i)~=0
        hist = X(i)*log2(X(i));
        entro = entro-hist;
     end
end
display(entro)
display(entro*length(y)/8000)
display(length(y)/1000)
grid on
```

شکل ۴: کدهای مربوط به گام چهارم:

گام چهارم:

پرسش اول:

مقدار مقدار انتروپی ۱۷.۹ بدست آمد و طبق قضیه ی اول شانون مرز فشرده سازی را میتوان با ضرب تعداد نمونه ها در مقدار انتروپی بدست آورد. پس فایل صوتی را تا تقریبا ۱۲۸۴ کیلوبایت میتوان فشرده سازی کرد، این در حالی ست که حجم فایل صوتی ما ۱۰۲۹ کیلوبایت است. نتیجه به دست آمده از قضیه ی شانون منطقی نمی باشد چرا که از طرفی انتروپی به دست آمده از فایل نمونه برداری شده به دست آمده و لزوما دقت لازم را ندارد و از طرفی دیگر احتمال ظاهر شدن سمبل های منبع در کنار یکدیگر نیز مستقل نیست و فرض مستقل بودن آنها نیز غلط است لذا این مرز دقیق نیست.

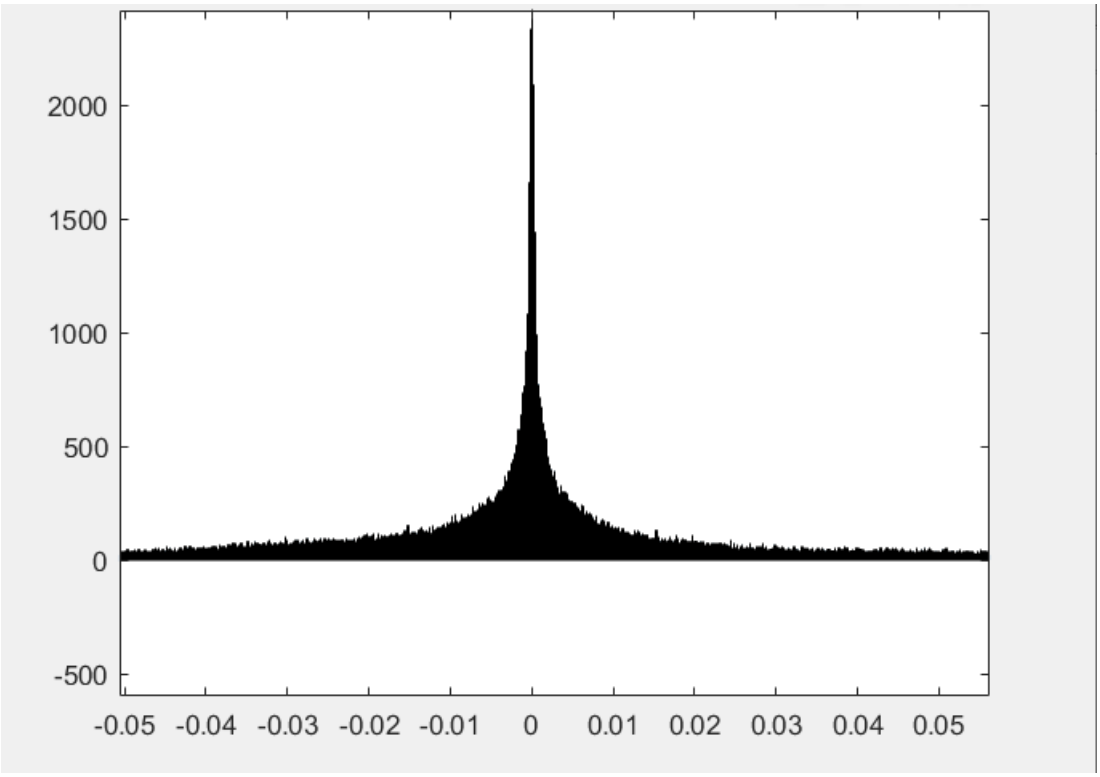
```
entro =

   9.1728e-05

    0.0030

  263.1680
```

شکل ۵: مقدار انتروپی



شکل ۶: نمودار هیستوگرام

پرسش دوم:

الگوریتم‌های فشرده‌سازی بهینه معمولاً فراوانی آماری را به طریقی به کار می‌گیرند که بتواند اطلاعات فرستنده را خلاصه‌تر و بدون خطا نمایش دهد. فشرده‌سازی بهینه امکان‌پذیر است چون اغلب اطلاعات جهان واقعی دارای فراوانی آماری هستند. برای مثال در زبان فارسی حرف "الف" خیلی بیشتر از حرف "ژ" استفاده می‌شود و احتمال اینکه مثلاً حرف "غین" بعد از حرف "ژ" بیاید بسیار کم است. نوع دیگری از فشرده‌سازی، که فشرده‌سازی بر اتلاف یا کدگذاری ادراکی نام دارد که در صورتی مفید است که درصدی از صحت اطلاعات کفایت کند. به‌طور کلی فشرده‌سازی بااتلاف توسط جستجو روی نحوه دریافت اطلاعات مورد نظر توسط افراد راهنمایی می‌شود. برای مثال، چشم انسان نسبت به تغییرات ظریف در روشنایی حساس تر از تغییرات در رنگ است. فشرده‌سازی تصویر به روش JPEG طوری عمل می‌کند که از بخشی از این اطلاعات کم ارزش تر "صرف نظر" می‌کند. فشرده‌سازی بااتلاف روشی را ارائه می‌کند که بتوان بیشترین صحت برای درصد فشرده‌سازی مورد نظر را به دست‌آورد. در برخی موارد فشرده‌سازی شفاف (نا محسوس) مورد نیاز است؛ در مواردی دیگر صحت قربانی می‌شود تا حجم اطلاعات تا حد ممکن کاهش بیابد. روش‌های فشرده‌سازی بهینه برگشت پذیرند به نحوی که اطلاعات اولیه قابلیت بازیابی به‌طور دقیق را دارند در حالی که روش‌های بااتلاف، از دست دادن مقداری از اطلاعات را برای دست یابی به فشردگی بیشتر می‌پذیرند. البته همواره برخی از داده ها وجود دارند که الگوریتم‌های فشرده‌سازی بهینه اطلاعات در فشرده‌سازی آن‌ها ناتوان هستند. در واقع هیچ الگوریتم فشرده‌سازی ای نمی‌تواند اطلاعاتی که هیچ الگوی قابل تشخیصی ندارند را فشرده‌سازی کند. بنابراین تلاش برای فشرده‌سازی اطلاعاتی که قبلاً فشرده‌شده‌اند معمولاً نتیجه عکس داشته (به جای کم کردن حجم، آن را زیاد می‌کند)، هم چنین است تلاش برای فشرده‌سازی هر اطلاعات کدشده‌ای (مگر حالتی که کد بسیار ابتدایی باشد). در عمل، فشرده‌سازی بااتلاف نیز به مرحله‌ای می‌رسد که فشرده‌سازی مجدد دیگر تأثیری ندارد، هرچند یک الگوریتم بسیار بااتلاف، مثلاً الگوریتمی که همواره بایت آخر فایل را حذف می‌کند، همیشه به مرحله‌ای می‌رسد که دیگر فایل تهی می‌شود

```
30      %Part5
31 -    uniq=unique(y);
32 -    uniq2 = zeros(1,length(uniq));
33 -    u = zeros(1,length(uniq)+1);
34 -    for i=1:length(uniq)
35 -        uniq2(i)=uniq(i) ;
36 -        u(i)=uniq(i);
37 -    end
38 -    u(length(uniq)+1)=2;
39 -    hist=histogram(y,u);
40 -    res=hist.Values./length(y);
41 -    dict=huffmandict(uniq2,res);
42 -    compressdata=huffmanenco(y,dict);
43 -    disp(length(compressdata)/8000)
44 -    disp(length(compressdata)/64000)
```

شکل ۷: کدهای مربوط به گام پنجم

گام پنجم:

برای ذخیره سازی فایل پس از کدگذاری هافمن به تقریبا ۱۴۰۰ کیلوبایت نیاز داریم که طبق نتیجه به دست آمده از قسمت چهارم کمی نزدیک به مرز شانون است. فایل اصلی دارای حجم ۱۰۲۹ کیلوبایت است و برای انتقال فایل در کانال مورد نظر به تقریبا ۱۲ ثانیه نیاز داریم.