

# Computer Assignment 6

مصطفی کرمانی نیا 810101575

امیر نداف فهمیده 810101540

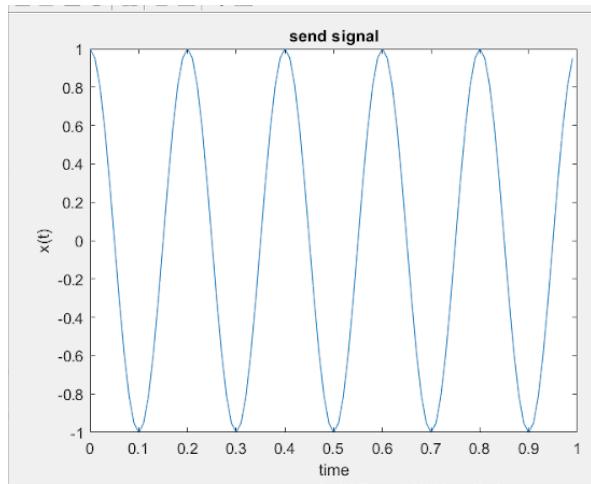
## بخش اول

### (1-1) تمرین

```
%% Part 1
fc = 5;
tstart = 0;
tend = 1;
fs = 100;

t = tstart: 1/fs : tend - 1/fs;
x = cos(2*pi*fc*t);
figure
plot(t, x)
xlabel('time');
ylabel('x(t)');
title('send signal');
```

ابتدا مقادیر گفته شده در سوال را ست می‌کنیم و  $t$  را همانطور که در صورت پروژه است قرار می‌دهیم سپس سیگنال ارسالی را به صورت  $\cos(2\pi f_c t)$  ایجاد می‌کنیم و رسم می‌کنیم. شکل سیگنال ارسالی به صورت زیر است.



## تمرین 1 (2-1)

```
%% Part 2

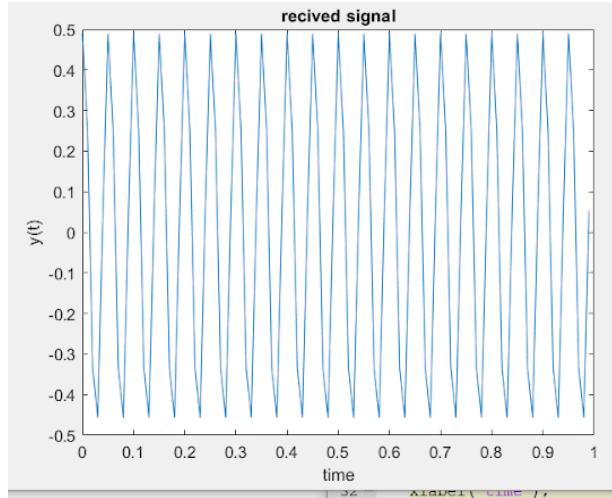
alpha = 0.5;
Beta = 0.3;
R = 250; %Km
V = 180; %Km / h
fd = Beta * V / 3.6;
c = 3*10^8;
td = 2/c * R * 1000;

y = alpha * cos(2*pi*(fc+fd)*(t-td));

figure
plot(t, y)
xlabel('time');
ylabel('y(t)');
title('received signal');
```

در این بخش دوباره مقادیر که در صورت سوال آمده را سمت می‌کنیم سپس بر اساس فرمول هایی که در صورت سوال آمده  $f_d$ ,  $t_d$  را محاسبه می‌کنیم (دقت شود که

مقادیر سرعت و فاصله را باید به ترتیب بر حسب متر بر ثانیه و متر در این محاسبات استفاده کنیم) و سیگنال دریافتی را همانطور که در صورت پژوهه آمده به صورت می‌سازیم و پلاس می‌کنیم. سیگنال دریافتی به شکل  $\alpha \cos(2\pi(f_c + f_d)(t - t_d))$  زیر است.



### تمرین 1 (3-1)

با توجه به راهنمایی، اگر ما بتوانیم  $f_{new}$ ,  $t_d$ ,  $\varphi$  را به دست آوریم می‌توانیم  $f_d$  را به دست آوریم و به کمک روابط بین این دو با  $R$  و  $V$ ، این دو را نیز به دست آوریم. برای به دست آوردن  $\varphi$  می‌توانیم با یک تبدیل فوریه گرفتن از سیگنال دریافتی، مقادیر این دو را به دست آوریم در ادامه در یک عکس حل دستی این تبدیل فوریه آمده است و نشان داده می‌شود که مقدار  $f$  برابر فرکانسی است که در تبدیل ما ضریب داریم و مقدار  $\varphi$  برابر فاز در این فرکانس است. حال به توضیح کد می‌پردازیم.

```

%% Part 3
N = (tend - tstart) * fs;
f = -fs/2 : fs/N : fs/2-fs/N;

FTR = fftshift(fft(y));
FTR = FTR/max(abs(FTR));
[value idx1] = max(FTR(51:100));
freq = f(50+idx1);

phaseVal = abs(angle(FTR(50+idx1)));

fdnew = freq - fc;
tdnew = phaseVal/(2*pi*(fc+fdnew));
Vnew = fdnew * 3.6/Beta;
Rnew = round(tdnew / 1000 * 0.5 * c);

disp(['fd = ', num2str(fdnew)]);
disp(['td = ', num2str(tdnew)]);
disp(['V = ', num2str(Vnew)]);
disp(['R = ', num2str(Rnew)]);

```

در این بخش ابتدا  $N$  را که تعداد سمپل ها است، به دست می آوریم سپس بازه فرکانس ها را مطابق هایلایت سبز در صورت پروژه تعریف می کنیم. سپس از سیگنال دریافتی تبدیل فوریه گرفته و آن را نرمالایز می کنیم. با توجه به تبدیل فوریه این سیگنال که به صورت دستی در عکس زیر حساب شده، دو ضریب متقارن داریم که در مثبت و منفی

$f_{new}$  اتفاق می افتد.

$$\begin{aligned}
x_{(+)} &= a \cos(2\pi f_{new} t + \varphi) \longrightarrow \hat{x}_{(w)} = \mathcal{F}(x_{(+)}) \\
\hat{x}_{(w)} &= a \mathcal{F}(\cos(2\pi f_{new} t + \varphi)) = a \mathcal{F}(\cos(2\pi f_{new}(t + \frac{\varphi}{2\pi f_{new}}))) \\
&= a e^{iw\frac{\varphi}{2\pi f_{new}}} \mathcal{F}(\cos(2\pi f_{new}t)) = a \mathcal{F} e^{iw\frac{\varphi}{2\pi f_{new}}} (\delta_{(w-2\pi f_{new})} + \delta_{(w+2\pi f_{new})}) \\
&= a \mathcal{F} \left( e^{i\varphi} \delta_{(w-2\pi f_{new})} + e^{-i\varphi} \delta_{(w+2\pi f_{new})} \right) \\
\longrightarrow \hat{x}_{(w)} &= a \mathcal{F} \left( e^{i\varphi} \delta_{(w-2\pi f_{new})} + e^{-i\varphi} \delta_{(w+2\pi f_{new})} \right)
\end{aligned}$$

حال که فرکانس را پیدا کردیم کافی است مقدار فاز را در این فرکانس به دست آوریم که طبق حل دستی برابر  $\varphi_{new}$  خواهد بود. با توجه به روابطی که در صورت سوال آمده می‌توان  $f_d$  را از فرمول های زیر به دست آورد.

$$f_d = f_{new} - f_c \quad , \quad t_d = \varphi_{new} / (2\pi f_{new}) = \varphi_{new} / (2\pi(f_c + f_d))$$

که در کد با ماکس گرفتن از تبدیل فوریه و پیدا کردن ضریب ها،  $f_{new}$  را پیدا کردیم و با یافتن فاز در این فرکانس  $\varphi_{new}$  را یافتیم و بعد با فرمول های بالا  $t_d$ ,  $f_d$  را به دست آوردیم. ) و در آخر با استفاده از روابط داده شده به سوال V و R را به صورت زیر به دست آوردیم (دقت شود که در این روابط Ma V و R را به کیلومتر بر ساعت و کیلومتر تبدیل کردیم.

$$V = 3.6 * f_d / \beta \quad , \quad R = 1000 * t_d * c/2$$

در آخر هم مقادیر محاسبه شده را گزارش کردیم که خروجی به صورت زیر است.

$f_d = 15$
$t_d = 0.0016667$
$V = 180$
$R = 250$

با توجه به مقادیر  $V$  و  $R$  به دست آمده می‌بینیم که روش ما به درستی کار می‌کند.

## تمرین (4-1)

```
59 -     std = 0.01;
60 -     noise = randn(1,length(y));
61 -     True_R = 1;
62 -     True_V = 1;
63 -     for i=1:1000
64 -         noise_i = std*noise;
65 -         y_noisy = y + noise_i;
66 -
67 -         FTR = fftshift(fft(y_noisy));
68 -         FTR = FTR/max(abs(FTR));
69 -         [value idx1] = max(FTR(51:100));
70 -         freq = f(50+idx1);
71 -         phaseVal = abs(angle(FTR(50+idx1)));
72 -
73 -         fdnew = freq - fc;
74 -         tdnew = phaseVal/(2*pi*(fc+fdnew));
75 -         Vnew = fdnew * 3.6/Beta;
76 -         Rnew = round(tdnew / 1000 * 0.5 * c);
77 -
78 -         if Rnew ~= R && True_R==1
79 -             disp(['With std = ', num2str(std), ' Rnew = ', num2str(Rnew), ' detected wrong']);
80 -             True_R=0;
81 -         end
82 -
83 -         if Vnew ~= V && True_V==1
84 -             disp(['With std = ', num2str(std), ' Vnew = ', num2str(Vnew), ' detected wrong']);
85 -             True_V=0;
86 -         end
87 -
88 -         if True_V + True_R == 0
89 -             break;
90 -         end
91 -         std = std + 0.01;
92 -     end
```

در این بخش ابتدا یک انحراف معیار برای نویز که از توزیع نرمال پیروی می‌کند تعریف می‌کنیم که برابر 0.01 گذاشتیم. سپس یک نویز مثل پروژه های قبلی تولید می‌کنیم ولی انحراف معیار که عملا قدرت نویز را مشخص می‌کند را فعلا در آن ضرب نمی‌کنیم. بعد دو تا فلگ  $True\_R$  و  $True\_V$  را به یک سمت کردیم تا اولین بار که آن را اشتباه محاسبه کردیم، آن را صفر کنیم، سپس یک حلقه 1000 تایی داریم و در هر مرحله  $std$  را در نویز ضرب می‌کنیم تا قدرت نویز را تنظیم کنیم سپس  $y$  تولید شده در بخش های قبل را با  $y\_noise$  جمع می‌کنیم تا سیگنال نویزی را بسازیم سپس با همان کد بخش قبل  $y$  و  $R$  را به دست می‌آوریم سپس چک می‌کنیم که اگر با  $R$  و  $V$  درست برابر نبود، قدرت نویز و مقدار اشتباه را پرینت می‌کنیم و فلگ مربوط را صفر می‌کنیم

در آخر هم اگر هر دو فلگ صفر بودند از حلقه می‌آییم ییرون در غیر این صورت 0.01 به std اضافه می‌کنیم و دوباره حلقه اجرا می‌شود تا نویز با قدرت بیشتر به سیگنال y اضافه شود. این حلقه تا جایی ادامه دارد که هر دو R و V یک بار اشتباه تشخیص داده شوند. دقیق شود که نتایج خروجی همواره یکسان نیس چون تابع Randn هر سری رندوم نویز درست می‌کند اما می‌توان با توجه به خروجی هایی که در زیر آمد، گفت که R نسبت به نویز خیلی حساس است و با نویز با قدرت کم (std=0.01) هم اشتباه تشخیص داده می‌شود ولی V به نسبت به نویز مقاوم تر است و در نویز با قدرت بالاتر اشتباه محاسبه می‌شود (std=1.11).

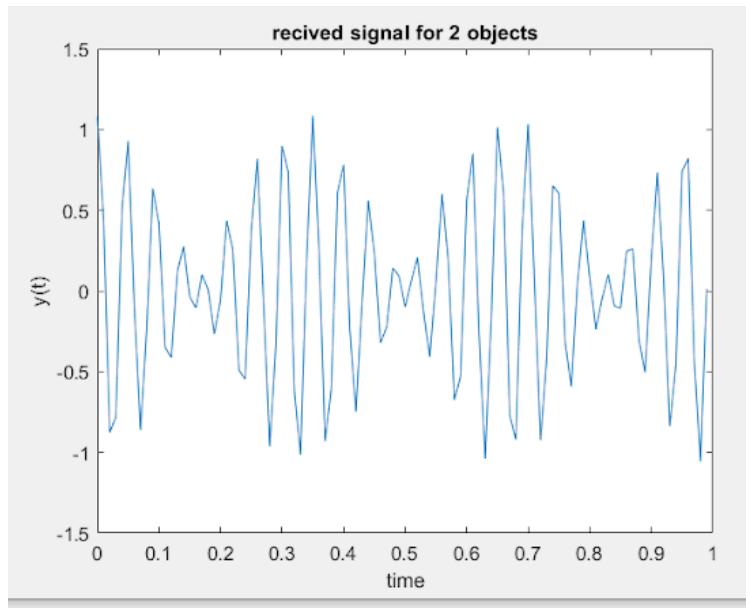
```
With std = 0.01Rnew = 245 detected wrong
With std = 1.11Vnew = 528 detected wrong
```

## (5-1) تمرین

```

94 %% Part 5
95
96 alpha1 = 0.5;
97 Beta = 0.3;
98 R1 = 250; %Km
99 V1 = 180; %Km / h
100 fd1 = Beta * V1 / 3.6;
101 tdl = 2/c * R1 * 1000;
102
103 alpha2 = 0.6;
104 R2 = 200; %Km
105 V2 = 216; %Km / h
106 fd2 = Beta * V2 / 3.6;
107 td2 = 2/c * R2 * 1000;
108
109 y1 = alpha1 * cos(2*pi*(fc+fd1)*(t-tdl));
110 y2 = alpha2 * cos(2*pi*(fc+fd2)*(t-td2));
111 y = y1 + y2;
112
113 figure
114 plot(t, y)
115 xlabel('time');
116 ylabel('y(t)');
117 title('recived signal for 2 objects');
```

در این بخش مانند بخش 1-2 مقادیر گفته شده در صورت سوال را ست کردیم و برای هر جسم سیگنالی که به سمت گیرنده می‌فرستند را تشکیل دادیم. سپس طبق صورت سوال این دو سیگنال را با هم جمع کردیم و سیگنال دریافتی حاصل را پلات کردیم. سیگنال دریافتی به صورت زیر است.



### تمرین 6-1

با توجه به این که تبدیل فوریه یک اپراتور خطی است، نتیجه تبدیل فوریه گرفتن از این سیگنال برابر با جمع تبدیل فوریه گرفتن از سیگنال اکو شده از هر جسم است پس انتظار داریم در فرکانس های مثبت این بار دو تا ضریب بینیم که هر کدام در نتیجه اکو شدن سیگنال از یکی از اجسام است. با این تفاسیر ما می‌توانیم  $f_{new}$  و  $\varphi_{new}$  را برای هر دو سیگنال اکو شده را توسط یک سیگنال دریافتی به دست آوریم و بر اساس همان روابط قبلی به  $V$ ,  $R$  هر کدام برسیم.

```

119 %% Part 6
120 FTR = fftshift(fft(y));
121 FTR = FTR/max(abs(FTR));
122
123 [value idx1] = max(FTR(51:100));
124 freq1 = f(50+idx1);
125 phaseVal1 = abs(angle(FTR(50+idx1)));
126 FTR(50+idx1) = 0;
127
128 [value idx2] = max(FTR(51:100));
129 freq2 = f(50+idx2);
130 phaseVal2 = abs(angle(FTR(50+idx2)));
131
132
133
134 fdnew1 = freq1 - fc;
135 tdnew1 = phaseVal1/(2*pi*(fc+fdnew1));
136 Vlnew = fdnew1 * 3.6/Beta;
137 Rlnew = tdnew1 / 1000 * 0.5 * c;
138
139 fdnew2 = freq2 - fc;
140 tdnew2 = phaseVal2/(2*pi*(fc+fdnew2));
141 V2new = fdnew2 * 3.6/Beta;
142 R2new = round(tdnew2 * c / 1000 * 0.5);
143
144 disp(['For object 1 V= ', num2str(Vlnew), 'and R= ', num2str(Rlnew)]);
145 disp(['For object 2 V= ', num2str(V2new), 'and R= ', num2str(R2new)]);
146

```

در این بخش از کد همانطور که بالا شرح داده شد ابتدا از سیگنال دریافتی، تبدیل فوریه می‌گیریم. سپس در بخش فرکانس‌های مثبت، به دنبال ماکزیمم می‌گردیم که در واقع  $f_{new}$  یک جسم برابر با فرکانسی است که این ماکزیمم در آن قرار دارد. سپس  $\varphi_{new}$  را نیز برای این جسم پیدا می‌کنیم که برابر با فاز در این فرکانس است. بعد از اینکه اطلاعات مورد نیاز برای محاسبه سرعت این جسم را استخراج کردیم مقدار تبدیل فوریه در این فرکانس را برابر صفر می‌گذاریم تا در واقع ضریب در این فرکانس را از بین ببریم. با توجه به توضیحات بالا این تبدیل فوریه یک ضریب دیگر هم دارد که شامل اطلاعات سیگنال اکو شده از جسم دوم است. برای اینکه ما این ضریب دوم را پیدا کنیم کافی است یک بار دیگر از تبدیل فوریه این سیگنال ماکزیمم بگیریم تا به فرکانسی که در آن ضریب ناشی از سیگنال اکو شده از جسم دوم است را بیابیم و سپس فاز در این فرکانس

برابر با  $\varphi$  این جسم است. حال که  $f_{new}$ ,  $V_{new}$  برای هر دو جسم را یافتیم به کمک روابطی که در بخش 3-1 آمده سرعت و فاصله هر کدام را استخراج می‌کنیم و آن را گزارش می‌کنیم. خروجی این بخش به صورت زیر است.

For object 1  $V = 216$  and  $R = 200$

For object 2  $V = 180$  and  $R = 250$

که همانطور که مشاهده می‌شود، مقادیر  $V$ ,  $R$  برای هر جسم به درستی استخراج شده پس کد این بخش به درستی کار می‌کند.

### (7-1) تمرین

خیر ، با توجه به نکته سوم در مقدمه، گام فرکانسی برای سیگنال دریافتی برابر  $\Delta_f = 1/T = 1Hz$ , ( $T = 1s$ ) آنها ضریب داریم در سیگنال های اکو شده توسط دو جسم 1 باشد تا بتوانیم این دو را تمایز دهیم. با توجه به اینکه فرکانس هایی که در آنها ضریب داریم با سرعت رابطه مستقیم دارند و از رابطه  $f_{new} = f_c + \beta * V/3.6$  به دست می‌آید، اگر سرعت هر دو جسم برابر شوند به جای اینکه دو ضریب داشته باشیم، یک ضریب در یک فرکانس داریم و این باعث می‌شود که نتوانیم سرعت جسم دوم را پیدا کنیم. همچنین چون ضریب ها در تبدیل فوریه روی یک فرکانس می‌افتد فاز های دو سیگنال با هم تداخل خواهند داشت و فاصله را نیز به درستی به دست نمی‌آوریم.

با توجه به گام فرکانسی، باید فاصله بین دو فرکانس هایی که ضریب اتفاق می‌افتد بیشتر از یک  $\beta * |V_1 - V_2| / 3.6$  شود. با توجه به رابطه بالا اگر اختلاف این دو فرکانس را بگیریم برابر  $3.6 * \beta$  باشد که اگر این مقدار را بزرگتر مساوی یک بگذاریم اختلاف سرعت باید بیشتر از  $3.6 * \beta$  باشد.

---

کیلومتر بر ساعت باشد که برای  $\beta = 0.3$  باید اختلاف سرعت ها بیشتر از 12 کیلومتر بر ساعت باشد.

$$\beta * |V_1 - V_2| / 3.6 \geq 1 \rightarrow |V_1 - V_2| \geq 3.6/\beta$$

برای اینکه این را چک کنیم هم کد بخش قبل را دوباره کپی کردم و اختلاف سرعت ها را 12 گذاشتم و دیدم به درستی فاصله و سرعت تخمین زده می شود.

---

```
148 %% Part 7 test our Idea
149
150 - alpha1 = 0.5;
151 - Beta = 0.3;
152 - R1 = 250; %Km
153 - V1 = 180; %Km / h
154 - fd1 = Beta * V1 / 3.6;
155 - tdl = 2/c * R1 * 1000;
156
157 - alpha2 = 0.6;
158 - R2 = 200; %Km
159 - V2 = 192; %Km / h
160 - fd2 = Beta * V2 / 3.6;
161 - td2 = 2/c * R2 * 1000;
162
163 - y1 = alpha1 * cos(2*pi*(fc+fd1)*(t-td1));
164 - y2 = alpha2 * cos(2*pi*(fc+fd2)*(t-td2));
165 - y = y1 + y2;
166
```

---

```

167 - FTR = fftshift(fft(y));
168 - FTR = FTR/max(abs(FTR));
169
170 - [value idx1] = max(FTR(51:100));
171 - freq1 = f(50+idx1);
172 - phaseVal1 = abs(angle(FTR(50+idx1)));
173 - FTR(50+idx1) = 0;
174
175 - [value idx2] = max(FTR(51:100));
176 - freq2 = f(50+idx2);
177 - phaseVal2 = abs(angle(FTR(50+idx2)));
178
179
180
181 - fdnew1 = freq1 - fc;
182 - tdnew1 = phaseVal1/(2*pi*(fc+fdnew1));
183 - Vlnew = fdnew1 * 3.6/Beta;
184 - Rlnew = tdnew1 / 1000 * 0.5 * c;
185
186 - fdnew2 = freq2 - fc;
187 - tdnew2 = phaseVal2/(2*pi*(fc+fdnew2));
188 - V2new = fdnew2 * 3.6/Beta;
189 - R2new = round(tdnew2 * c / 1000 * 0.5);
190
191 - disp('test that if |V1-V2|>= 12 it works correctly');
192 - disp(['For object 1 V= ', num2str(Vlnew), 'and R= ', num2str(Rlnew)]);
193 - disp(['For object 2 V= ', num2str(V2new), 'and R= ', num2str(R2new)]);
194

```

```

test that if |V1-V2|>= 12 it works correctly
For object 1 V= 192and R= 200
For object 2 V= 180and R= 250
```

```

## (8-1) تمرین

بله زیرا فاصله مربوط به فاز در دو فرکانسی که ضریب داریم می‌شود و اگر فاز در این دو فرکانس برابر باشد، سرعت دو جسم برابر می‌شود ولی چون در دو فاز مختلف قرار دارند (اگر شرایط اختلاف سرعت بخش قبل برقرار باشد)، با هم تداخل ندارند و فاصله هر جسم به درستی به دست می‌آید.

---

## تمرین 1 (9-1)

اگر فاصله بین سرعت های اجسام در شرایط بخش 7-1 صدق کند در تبدیل فوریه سیگنال دریافتی، ما به تعداد اجسام، ضریه خواهیم داشت که با به دست آوردن فرکانس هایی که در آن ها ضریه داریم و فاز در آن فرکانس ها می توان سرعت و فاصله را برای هر جسم پیدا کرد. (هر فرکانس که در آن ضریه داریم مربوط به یک جسم می شود).

### بخش دوم

|   |           |   |             |
|---|-----------|---|-------------|
| C | 523.25 Hz | C | # 554.37 Hz |
| D | 587.33 Hz | D | # 622.25 Hz |
| E | 659.25 Hz |   |             |
| F | 698.46 Hz | F | # 739.99 Hz |
| G | 783.99 Hz | G | # 830.61 Hz |
| A | 880 Hz    | A | # 932.33 Hz |
| B | 987.77 Hz |   |             |

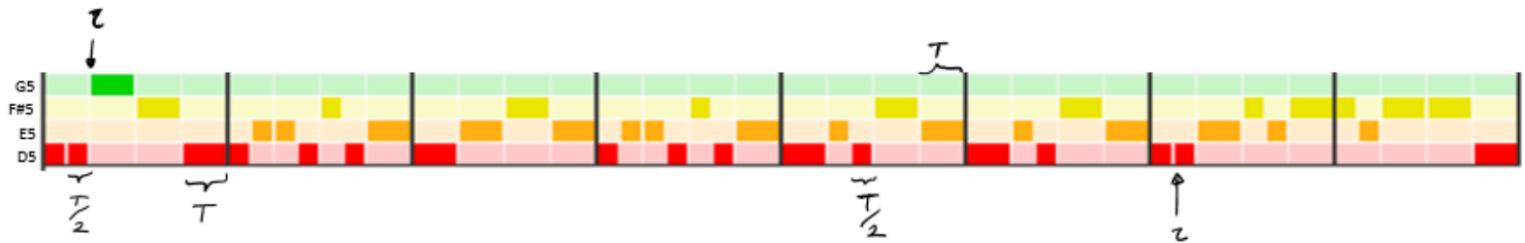
## تمرین 2 (1-2)

## نکته‌ی موسیقایی :

- هر کدام از حروف انگلیسی ای که نوشته شده اند، فقط نام یک نت است و هیچ اطلاعاتی درباره فرکانس دقیق یا اکتاو نمی‌دهد و مثلا G می‌تواند در هر اکتاو باشد: که هر نسخه از G فرکانس متفاوتی دارد.

- اما با توجه به این فرکانس‌های نوشته شده و تصویر ادیت شده، این فرکانس‌ها همگی متعلق به اکتاو پنجم هستند. یعنی به ترتیب C5, C#5, ..., A#5, B5 را نشان می‌دهند.

حالا با توجه به موسیقی ای که به ما داده شده است، داریم:



Bar1 : D5 (T/2) - D5 (T/2) - G5 (T) - F#5 (T) - D5 (T)

Bar2 : D5 (T/2) - E5 (T/2) - E5 (T/2) - D5 (T/2) - F#5 (T/2) - D5 (T/2) - E5 (T)

Bar3 : D5 (T) - E5 (T) - F#5 (T) - E5 (T)

Bar4 : D5 (T/2) - E5 (T/2) - E5 (T/2) - D5 (T/2) - F#5 (T/2) - D5 (T/2) - E5 (T)

Bar5 : D5 (T) - E5 (T/2) - D5 (T/2) - F#5 (T) - E5 (T)

Bar6 : D5 (T) - E5 (T/2) - D5 (T/2) - F#5 (T) - E5 (T)

Bar7 : D5 (T/2) - D5 (T/2) - E5 (T) - F#5 (T/2) - E5 (T/2) - F#5 (T)

Bar8 : F#5 (T/2) - E5 (T/2) - F#5 (T) - F#5 (T) - D5 (T)

---

## توضیح کد زده شده:

```
%% Part 1: Parameters
fs = 8000;
ts = 1/fs;
T = 0.5;
T_half = T / 2;
tau = 0.025; % Time gap between notes

% Frequencies of music keys that are used in this part
frequencies = {'D5', 587.33; 'E5', 659.25; 'F#5', 739.99; 'G5', 783.99};

t_full = 0:ts:T-ts;
t_half = 0:ts:T_half-ts;
silence = zeros(1, tau/ts); % Silence for gaps

final_signal = [];
```

- در مرحله‌ی اول پارامترهای و بردار‌های لازم را می‌سازیم، مثلاً فرکانس نمونه برداری را 8000 ست کرده و فاصله‌ی بین هر دو نمونه  $1/8000$  می‌شود.
- مدت زمان استاندارد فشردن هر کلید و مدت نصف آن که برای برخی نت‌ها استفاده شده را تعریف می‌کنیم.
- در نهایت هم مدت زمان سکوت بین هر دو کلید را تنظیم می‌کنیم.
- حالا یک بردار از `cell`‌ها می‌سازیم که 2 در 4 است و در آن نام هر نت به فرکانسش مپ می‌شود. و 4 نتی که در این پروژه استفاده کردیم را در آن قرار میدهیم

- حالا وکتور های زمانی را میسازیم. با همان فاصله نمونه برداری  $ts$  سه وکتور میسازیم که یکی برای زمان فشردن کامل یک کلید است، یکی برای زمان فشردن نصفه ی یک کلید است و یک وکتور صفر (سکوت) به طول تعداد سمپل های گرفته شده در زمان تاو میسازیم که بعدا به انتهای تک تک نت های نواخته شده اضافه می کنیم.

- نهایتا هم یک وکتور میسازیم که قرار است آهنگ نهایی را نت به نت داخلش قرار داده و بسازیم.

- نتیجه نهایی:

|              |               |
|--------------|---------------|
| final_signal | []            |
| frequencies  | 4x2 cell      |
| fs           | 8000          |
| silence      | 1x200 double  |
| T            | 0.5000        |
| t_full       | 1x4000 double |
| t_half       | 1x2000 double |
| T_half       | 0.2500        |
| tau          | 0.0250        |
| ts           | 1.2500e-04    |

```
%% Part 2: Helper Function to append a note with specified duration
function note_signal = append_note(note, duration, frequencies, silence)
    freq = frequencies{strcmp(frequencies(:,1), note), 2};
    note_signal = [sin(2*pi*freq*duration), silence];
end
```

- حالا برای افزودن هر نت به سیگنال نهایی، از یک تابع کمکی استفاده می کنیم که نام نت و وکتور زمان آن و آرایه شامل سلول های فرکانس ها و مدت سکوت بعد از هر

---

نت را گرفته و ابتدا فرکانس متناظر با نت ورودی را یافته و سپس سیگنال سینوسی متناظر با آن را میسازد نهایتا یک سیگنال شامل نت نواخته شده و سکوت انتهاش برمیگرداند.

- برای پیدا کردن فرکانس متناظر با یک نت به این روش عمل میکند که ابتدا در بین عناصر ستون اول سلول که همان نام نت ها هست، میگردد تا نام آن نت را پیدا کند و سپس عنصر متناظر با همان نام را که همان فرکانس است از سلول برداشته و در freq مینویسد.
- نحوه‌ی ساخت سیگنال نهایی هم که طبق دستور سوال است و یک سیگنال تک تن با طول زمانی و فرکانس مشخص تولید میکنیم.

```
% Part 3: Build the Song
% Bar 1
final_signal = [final_signal, ...
    append_note('D5', t_half, frequencies, silence), append_note('D5', t_half, frequencies, silence), ...
    append_note('G5', t_full, frequencies, silence), append_note('F#5', t_full, frequencies, silence), ...
    append_note('D5', t_full, frequencies, silence)];
```

```
% Bar 2
final_signal = [final_signal, ...
    append_note('D5', t_half, frequencies, silence), append_note('E5', t_half, frequencies, silence), ...
    append_note('E5', t_half, frequencies, silence), append_note('D5', t_half, frequencies, silence), ...
    append_note('F#5', t_half, frequencies, silence), append_note('D5', t_half, frequencies, silence), ...
    append_note('E5', t_full, frequencies, silence)];
```

```
% Bar 3
final_signal = [final_signal, ...
    append_note('D5', t_full, frequencies, silence), append_note('E5', t_full, frequencies, silence), ...
    append_note('F#5', t_full, frequencies, silence), append_note('E5', t_full, frequencies, silence)];
```

```
% Bar 4
final_signal = [final_signal, ...
    append_note('D5', t_half, frequencies, silence), append_note('E5', t_half, frequencies, silence), ...
    append_note('E5', t_half, frequencies, silence), append_note('D5', t_half, frequencies, silence), ...
    append_note('F#5', t_half, frequencies, silence), append_note('D5', t_half, frequencies, silence), ...
    append_note('E5', t_full, frequencies, silence)];
```

```
% Bar 5
final_signal = [final_signal, ...
    append_note('D5', t_full, frequencies, silence), append_note('E5', t_half, frequencies, silence), ...
    append_note('D5', t_half, frequencies, silence), append_note('F#5', t_full, frequencies, silence), ...
    append_note('E5', t_full, frequencies, silence)];
```

```
% Bar 6
final_signal = [final_signal, ...
    append_note('D5', t_full, frequencies, silence), append_note('E5', t_half, frequencies, silence), ...
    append_note('D5', t_half, frequencies, silence), append_note('F#5', t_full, frequencies, silence), ...
    append_note('E5', t_full, frequencies, silence)];

% Bar 7
final_signal = [final_signal, ...
    append_note('D5', t_half, frequencies, silence), append_note('D5', t_half, frequencies, silence), ...
    append_note('E5', t_full, frequencies, silence), append_note('F#5', t_half, frequencies, silence), ...
    append_note('E5', t_half, frequencies, silence), append_note('F#5', t_full, frequencies, silence)];

% Bar 8
final_signal = [final_signal, ...
    append_note('F#5', t_half, frequencies, silence), append_note('E5', t_half, frequencies, silence), ...
    append_note('F#5', t_full, frequencies, silence), append_note('F#5', t_full, frequencies, silence), ...
    append_note('D5', t_full, frequencies, silence)];
```

- در نهایت هم با توجه به نقشه‌ی داده شده و تابع کمکی ساخته شده، آهنگ را نت به نت به سیگنال نهایی می‌افزاییم و میزان به میزان این کار را میکنیم تا آهنگ نهایی تشکیل شود.

---

#### %% Part 4: Play the Song

```
sound(final_signal, fs);
```

- بعد از اتمام ساخت آهنگ، آنرا با تابع sound پخش میکنیم و میبینیم که طول و صدای نت‌ها کاملاً با نقشه‌ی اولیه همخوانی داشته است.

## (2-2) تمرین

- کدها کاملاً شبیه بخش اول بود و صرفاً یکسری نت دیگر اضافه کرده و استفاده کردیم و نهایتاً هم برای سیو کردن فایل صوتی از دستور audiowrite استفاده کردیم.

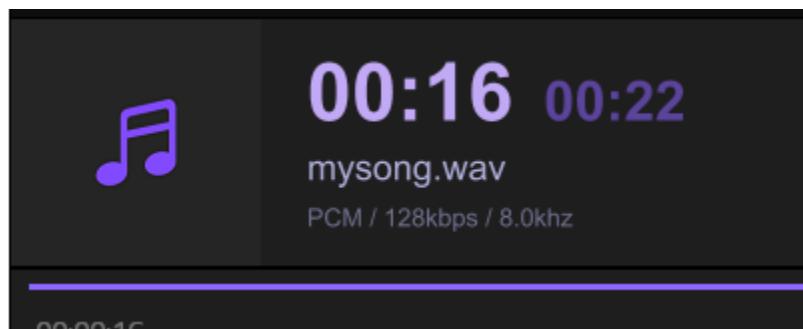
```
% Frequencies of music keys for "Canon in D" (Only Fifth Octave)
frequencies = {'D5', 587.33; 'A5', 880.00; 'B5', 987.77; 'F#5', 739.99; 'G5', 783.99};
```

```
%% Part 4: Play the Song and save it
%sound(final_signal, fs);
audiowrite('mysong.wav', final_signal, fs);
```

نهایتاً مقدار متغیرها به شرح زیر است:

| Name           | Type            |
|----------------|-----------------|
| final_signal   | 1x179200 double |
| {} frequencies | 5x2 cell        |
| fs             | 8000            |
| silence        | 1x200 double    |
| T              | 0.5000          |
| t_full         | 1x4000 double   |
| t_half         | 1x2000 double   |
| T_half         | 0.2500          |
| tau            | 0.0250          |
| ts             | 1.2500e-04      |

- همانطور که مشخص است سیگنال نهایی شامل 179200 سمپل است و چون فرکانس نمونه برداری را 8000 هرتز گرفتیم، فایل نهایی طبق انتظارمان 22.4 ثانیه خواهد بود:



- از طرفی سایز نهایی فایل صوتی 350 کیلوبایت است:

Size: 350 KB

- پس 179200 سمپل با 350 کیلوبایت ذخیره شده است که یعنی هر سمپل حدودا 1.953125 بایت اشغال کرده که معادل حدود 15.625 بیت می شود.  
اگر از لحاظ زمانی هم بررسی کنیم 350 کیلوبایت داریم و فایلمان 22.4 ثانیه است پس هر ثانیه 15625 بایت را اشغال کرده است.

---

## (3-2) تمرین

ایده‌ی ابتدایی و کلی:

- فایل ورودی، طبق فرضیات همین بخش تولید شده است، پس بین هر دو دفعه‌ای که کلید زده شده است، یک مقداری سکوت وجود دارد
- ما کد flexible ای زدیم و طبق آن لازم نیست طول مدت زمان نواختن هر نت لزوماً دو مقدار خاص 0.5 و 0.25 باشد یا سکوت‌های آن لزوماً با زمان مساوی یا مشخص انجام شوند.
- الگوریتم کلی به این صورت است که با توجه به سکوتی که بین هر دو نت وجود دارد، ما بباییم و این فایل را بخش بخش تقسیم کنیم و هر بخش شامل یک نت خواهد بود
- حالا از هر بخش تبدیل فوریه گرفته و سپس فرکانسی که در آن بخش مقدارش از بقیه بیشتر بوده است را برداشته و آن را با فرکانس‌های مختلف که لیبل خورده اند و نت معادل آن‌ها را می‌دانیم، مقایسه می‌کنیم.
- پس نتی که کمترین اختلاف را با فرکانس غالب آن بخش داشت، بعنوان نت آن بخش معرفی و چاپ می‌کنیم.

## توضیح کد زده شده:

```
%& Part 1: Load Audio and Parameters
[file, path] = uigetfile('*.*', 'Select an Audio File');
if isequal(file, 0)
    disp('No file selected.');
    return;
end
filePath = fullfile(path, file);
[audioSignal, fs] = audioread(filePath);
signalLength = length(audioSignal);
ts = 1/fs;
timeVector = (0:signalLength-1) * ts;

% Notes and Frequencies (Combined in a Cell Array)
notesData = {
    'C5', 523.25; 'C#5', 554.37; 'D5', 587.33; 'D#5', 622.25; 'E5', 659.25;
    'F5', 698.46; 'F#5', 739.99; 'G5', 783.99; 'G#5', 830.61; 'A5', 880.00;
    'A#5', 932.33; 'B5', 987.77
};
```

- این کد ابتدا یک پنجره باز می‌کند تا کاربر یک فایل صوتی با فرمت WAV را انتخاب کند. سپس بررسی می‌کند که آیا کاربر فایلی انتخاب کرده است یا خیر. اگر کاربر هیچ فایلی انتخاب نکرده باشد، پیغام "No file selected" نمایش داده شده و برنامه متوقف می‌شود.

- سپس مسیر کامل فایل انتخابی را ایجاد می‌کند و فایل صوتی را به صورت داده‌های نمونه (در آرایه audioSignal) و نرخ نمونه‌برداری (fs) می‌خواند.

- حالا تعداد کل نمونه‌های موجود در فایل صوتی (طول سیگنال) را ذخیره می‌کند. و بردار زمانی برای سیگنال ایجاد می‌کند. (هر مقدار در این بردار زمان متناظر با یک نمونه از سیگنال صوتی است).

- نهایتاً یک آرایه سلولی شامل نام نت‌ها (C5, D5, ...) و فرکانس متناظر هر نت (بر حسب هرتز) می‌سازد بنام .notesData.

---

```

%% Part 2: Extract Segments
startIndices = [];
endIndices = [];
for i = 1:signalLength-1
    if audioSignal(i) == 0 && audioSignal(i+1) ~= 0
        startIndices = [startIndices, i];
    end
    if i > 1 && audioSignal(i) == 0 && audioSignal(i-1) ~= 0
        endIndices = [endIndices, i-1];
    end
end

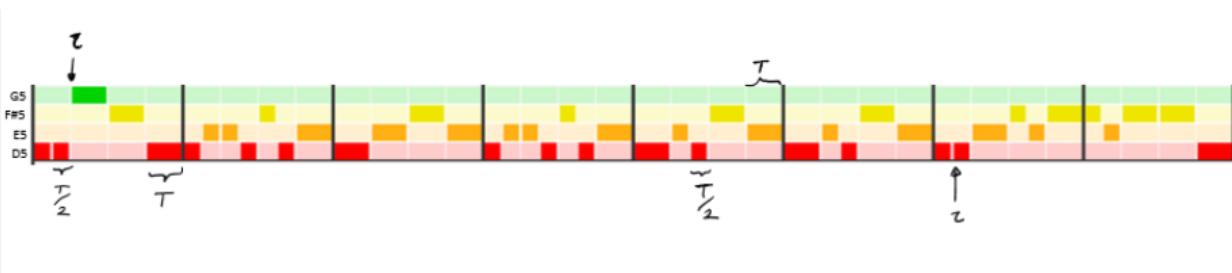
```

- دو آرایه برای ذخیره نقاط شروع و پایان هر بخش غیرساکت سیگنال (بخش‌هایی که مقدارشان صفر نیست) می‌سازیم.

- سپس برای پیمایش نمونه‌های سیگنال به منظور شناسایی نقاط شروع و پایان بخش‌های غیرساکت یک حلقه می‌زنیم و اگر مقدار فعلی صفر باشد و مقدار نمونه بعدی غیر صفر باشد، startIndices این به معنی شروع یک بخش غیرساکت است. پس اندیس نمونه فعلی در آرایه startIndices ذخیره می‌شود و اگر مقدار فعلی صفر باشد و مقدار نمونه قبلی غیر صفر باشد، این به معنی پایان یک بخش غیرساکت است. اندیس نمونه فعلی در آرایه endIndices ذخیره می‌شود.

- نهایتاً همانطور که انتظار میرفت، 44 تا بخش شناسایی شدند که مطابق با 44 تا نقطه بود که در بخش اول نواخته شد:

|  |              |                         |
|--|--------------|-------------------------|
|  | endIndices   | 1x44 double             |
|  | file         | 'CA6_song.wav'          |
|  | filePath     | 'D:\All\uni\semester 1' |
|  | fs           | 8000                    |
|  | i            | 136799                  |
|  | notesData    | 12x2 cell               |
|  | path         | 'D:\All\uni\semester 1' |
|  | signalLength | 136800                  |
|  | startIndices | 1x44 double             |
|  | ...          | ...                     |



```

%% Part 3: Analyze Segments
frequencies = [];
durations = [];
detectedNotes = {};
for i = 1:length(startIndices)
    segment = audioSignal(startIndices(i):endIndices(i));
    duration = (endIndices(i) - startIndices(i) + 1) / fs;
    durations = [durations, duration];

    Y = fftshift(fft(segment));
    Y = abs(Y) / max(abs(Y)); % Normalize FFT
    f = linspace(-fs/2, fs/2, length(Y)); % Frequency vector

    % Find Dominant Frequency
    [~, idx] = max(Y);
    dominantFreq = abs(f(idx));

    % Find Closest Note
    [~, noteIdx] = min(abs(cell2mat(notesData(:, 2)) - dominantFreq));
    detectedNote = notesData{noteIdx, 1};
    frequencies = [frequencies, notesData{noteIdx, 2}];
    detectedNotes = [detectedNotes, detectedNote];

    % Print each note with duration
    fprintf('Note: %s, Frequency: %.2f Hz, Duration: %.2f seconds\n', detectedNote, dominantFreq, duration);
end

```

- سه آرایه برای ذخیره اطلاعات می سازیم که در آن ها frequencies فرکانس غالب هر بخش و
- durations مدت زمان هر بخش و detectedNotes نت های موسیقی متضاظر هستند.
- سپس حلقه ای برای پیمایش هر بخش شناسایی شده میزیم و برای هر segment :سپس حلقه ای برای پیمایش هر بخش شناسایی شده میزیم و برای هر segment
- بخش صوتی بین نقاط شروع و پایان استخراج می شود، مدت زمان هر بخش بر حسب ثانیه محاسبه می شود و در آرایه durations ذخیره می شود.
- مدت زمان هر بخش بر حسب ثانیه محاسبه می شود و در آرایه durations ذخیره می شود.

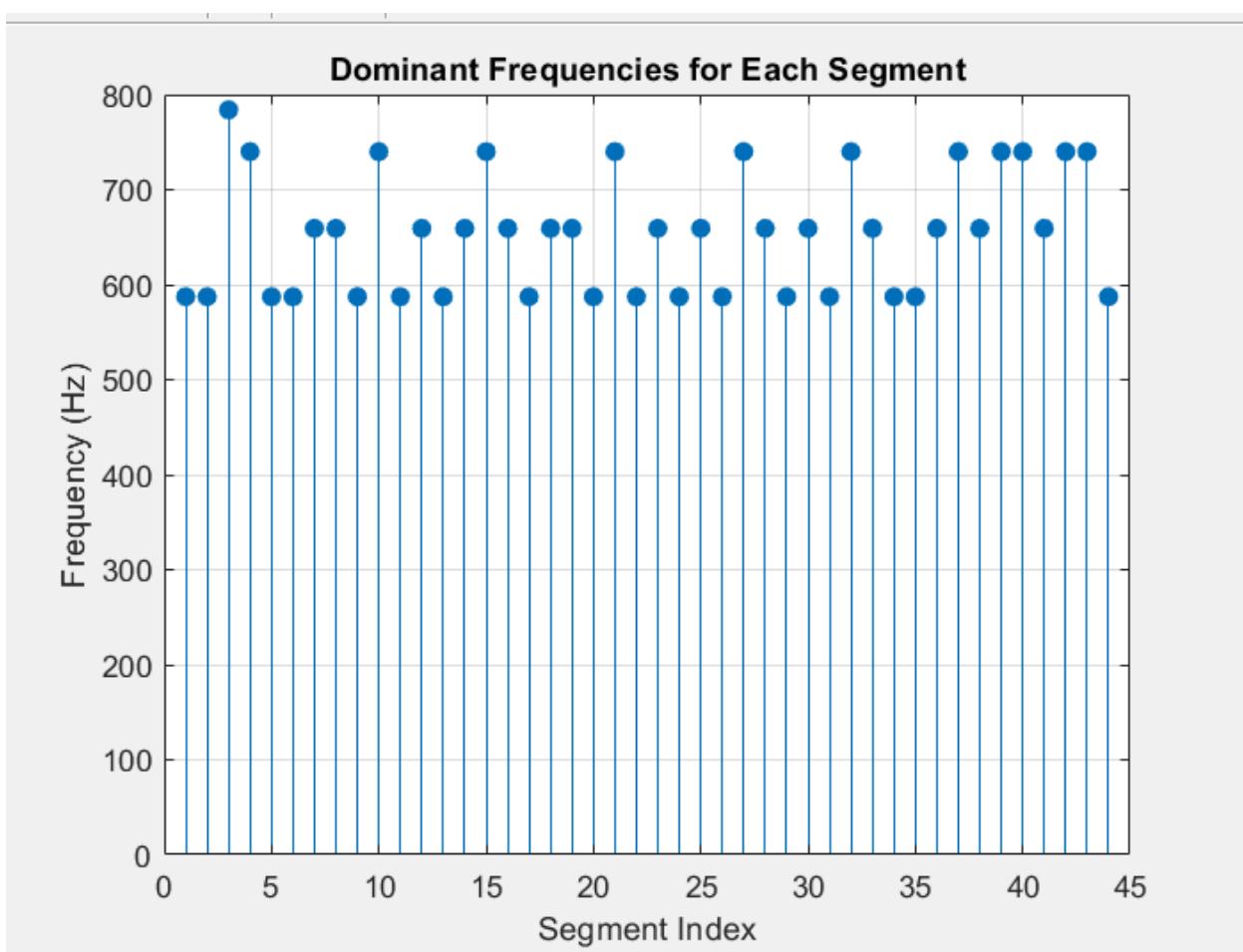
- 
- تبدیل فوریه روی بخش صوتی اعمال می شود تا فرکانس های موجود تحلیل شوند. برای fftshift
  - تنظیم محور فرکانس استفاده می شود و مقادیر FFT نرمال سازی می شوند تا مقایسه مقادیر ساده تر شود.
  - بردار فرکانس متناسب با نتایج FFT ساخته می شود.
  - بیشینه مقدار در نتایج FFT پیدا می شود که نشان دهنده فرکانس غالب است. و فرکانس غالب از روی مقدار اندیس بیشینه استخراج می شود.
  - فرکانس غالب با فرکانس های موجود در notesData مقایسه می شود تا نزدیک ترین فرکانس پیدا شود و نام نت متناظر با فرکانس غالب شناسایی می شود و فرکانس شناسایی شده در آرایه detectedNotes ذخیره frequencies می شود.
  - نهایتا اطلاعات نت (نام، فرکانس، مدت زمان) در کنسول چاپ می شود.

---

```
%> Part 4: Display Results plot
figure;
stem(1:length(frequencies), frequencies, 'filled');
xlabel('Segment Index');
ylabel('Frequency (Hz)');
title('Dominant Frequencies for Each Segment');
grid on;
```

- نهایتا برای بهتر دیده شدن نت های یافت شده، یک نمودار میله ای برای فرکانس های غالب رسم می کند. محور افقی نمایانگر شماره بخش ها و محور عمودی فرکانس ها است.

نهایتا هم با ران کردن کد به همچین نتیجه ای میرسیم:

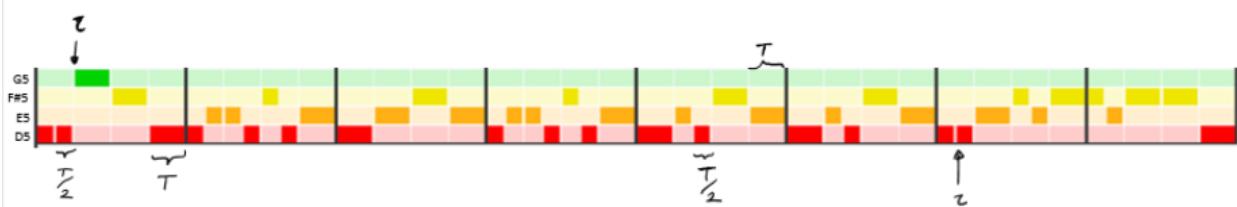


---

Note: D5, Frequency: 586.29 Hz, Duration: 0.25 seconds  
Note: D5, Frequency: 586.29 Hz, Duration: 0.25 seconds  
Note: G5, Frequency: 783.20 Hz, Duration: 0.50 seconds  
Note: F#5, Frequency: 739.18 Hz, Duration: 0.50 seconds  
Note: D5, Frequency: 587.15 Hz, Duration: 0.50 seconds  
Note: D5, Frequency: 586.29 Hz, Duration: 0.25 seconds  
Note: E5, Frequency: 658.33 Hz, Duration: 0.25 seconds  
Note: E5, Frequency: 658.33 Hz, Duration: 0.25 seconds  
Note: D5, Frequency: 586.29 Hz, Duration: 0.25 seconds  
Note: F#5, Frequency: 738.37 Hz, Duration: 0.25 seconds  
Note: D5, Frequency: 586.29 Hz, Duration: 0.25 seconds  
Note: E5, Frequency: 659.16 Hz, Duration: 0.50 seconds  
Note: D5, Frequency: 587.15 Hz, Duration: 0.50 seconds  
Note: E5, Frequency: 659.16 Hz, Duration: 0.50 seconds  
Note: F#5, Frequency: 739.18 Hz, Duration: 0.50 seconds  
Note: E5, Frequency: 659.16 Hz, Duration: 0.50 seconds  
Note: D5, Frequency: 586.29 Hz, Duration: 0.25 seconds  
Note: E5, Frequency: 658.33 Hz, Duration: 0.25 seconds  
Note: D5, Frequency: 586.29 Hz, Duration: 0.25 seconds  
Note: F#5, Frequency: 738.37 Hz, Duration: 0.25 seconds  
Note: D5, Frequency: 586.29 Hz, Duration: 0.25 seconds  
Note: E5, Frequency: 659.16 Hz, Duration: 0.50 seconds  
Note: D5, Frequency: 587.15 Hz, Duration: 0.50 seconds  
Note: E5, Frequency: 658.33 Hz, Duration: 0.25 seconds  
Note: D5, Frequency: 586.29 Hz, Duration: 0.25 seconds  
Note: F#5, Frequency: 739.18 Hz, Duration: 0.50 seconds  
Note: E5, Frequency: 659.16 Hz, Duration: 0.50 seconds  
Note: D5, Frequency: 586.29 Hz, Duration: 0.25 seconds  
Note: D5, Frequency: 586.29 Hz, Duration: 0.25 seconds

---

که دقیقاً با نت‌های نواخته شده و مدت زمان اجرای آنها مطابقت دارد:



- پس کد ما به درستی روی آهنگ ساخته شده در بخش اول، کار کرده و نت‌ها را تشخیص می‌دهد.