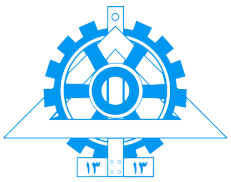


تمرین کامپیوتری ششم

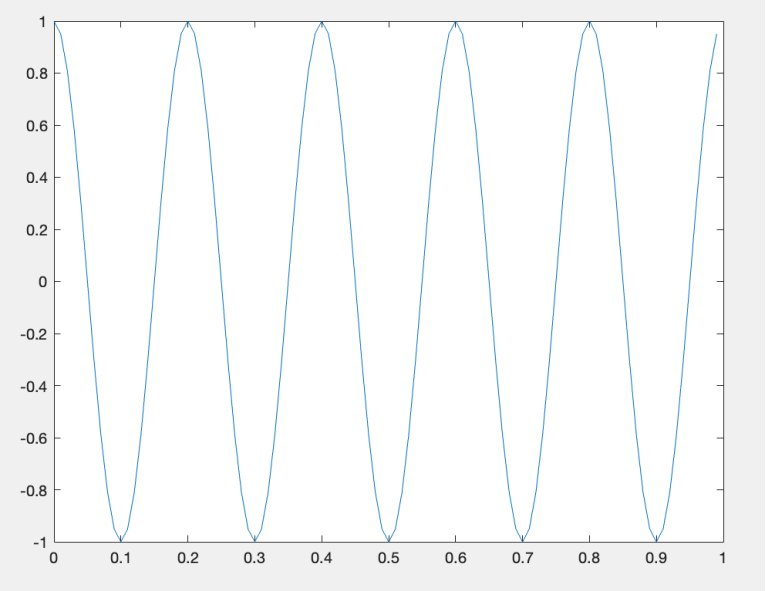
📡 سیگنال ها و سیستم ها 📡

مجید صادقی نژاد - 810101459
آرمان خورشیدی - 810101417
پاییز ۱۴۰۳
استاد: دکتر اخوان

بخش اول

تمرین ۱

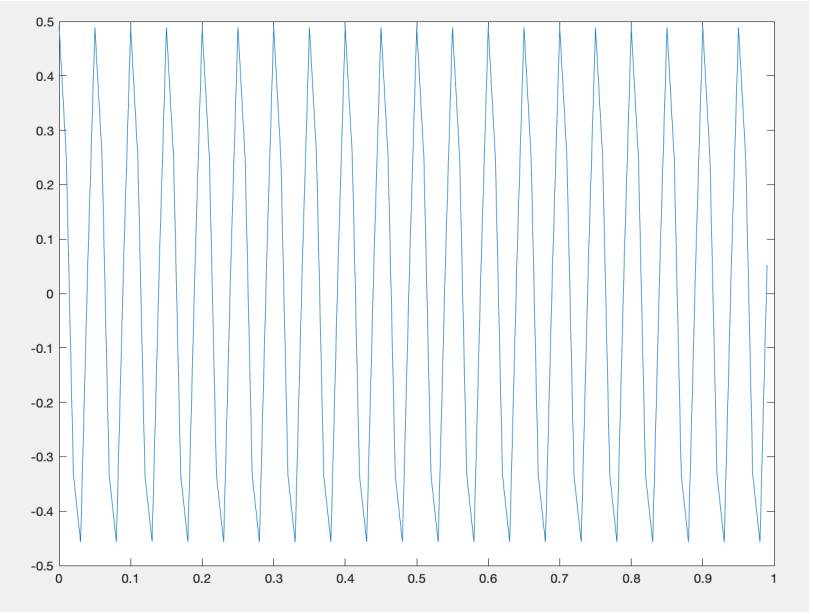
همانطور که در دستور پروژه خواسته شده بود سیگنال مورد نظر رسم می شود:




```
fs = 100;
ts = 1/fs;
t = 0:ts:1-ts;
fc = 5;
w = 2*pi*fc;
plot(t,cos(w*t));
```

تمرین ۲

مجددا خواسته ی دستور کار رسم می گردد:




```
t_start = 0;
t_end = 1;
fc = 5;
fs = 100;
V = 180;
R = 250;
B = 0.3;
A = 0.5;
C = 3e8;
p = 2/C;

V = V / 3.6;
R = R * 1000;
ts = 1/fs;
t = t_start:ts:t_end-ts;

signal = A*cos(2*pi*(fc+B*V)*(t-p*R));

plot(t,signal)
```

تمرین ۳

روش کار به این صورت است که ابتدا با توجه به رزولوشن فرکانسی محور فرکانسی را تشکیل می دهیم سپس از سیگنال دریافتی تبدیل فوریه می گیریم و مقدار فاز و اندازه ی آن را جدا می کنیم حال فرکانس های منفی را حذف می کنیم زیرا نیازی به آن ها نداریم حال فرکانسی که بیشترین اندازه را دارد میابیم، این فرکانس همان فرکانس جدید است که به دنبال آن هستیم و از آن جا که این فرکانس جدید حاصل مجموع فرکانس c و فرکانس داپلر است و ما فرکانس c را داریم به راحتی می توانیم فرکانس داپلر را از آن استخراج کنیم و سپس با استفاده از فرکانس داپلر و مقدار ثابت B مجددا به سادگی می توانیم مقدار V را به دست آوریم: $fd = B * V$ همچنین با داشتن فاز فرکانس اصلی می توان از رابطه ی $\phi = 2\pi f_{new} t_d$ به راحتی مقدار t_d را استخراج کرد. و سپس به وسیله ی آن و مقدار ثابت ρ مقدار R را حساب کرد $R\rho = t_d$ در نهایت عبارت های محاسبه شده به شکل زیر هستند:

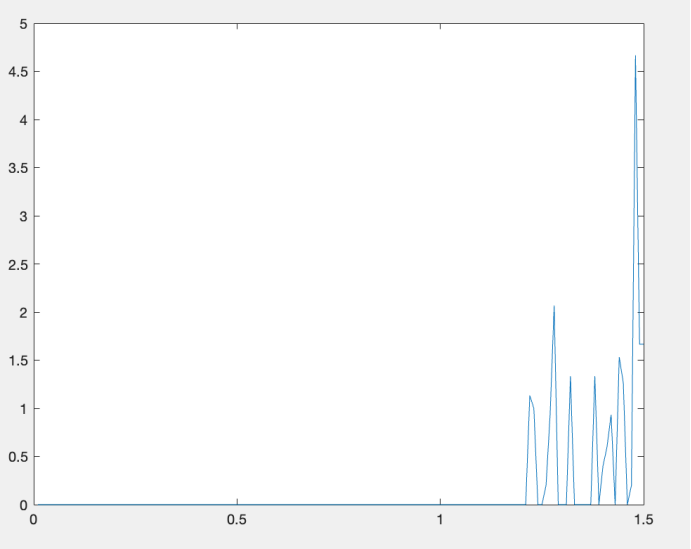
```
V =

    50


R =

   2.5000e+05
```

که دقیقا مطابق انتظار هستند کد تمامی مراحل بالا که توضیح داده شد به شکل زیر است:

```
resol = 1/(t_end-t_start);
freq = -fs/2:resol:fs/2-resol;
FT = fftshift(fft(signal));

absolute = abs(FT);
phase = angle(FT);

absolute = absolute(freq>=0);
phase = phase(freq>=0);
freq = freq(freq>=0);

[~,f_new] = max(absolute);
t_d = abs(phase(f_new)/(2*pi*freq(f_new)));
f_d = freq(f_new)-fc;

V = f_d/B
R = t_d/p
```

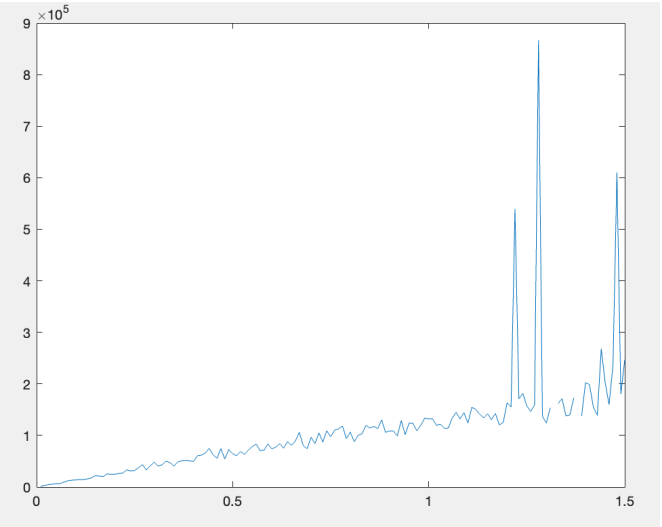
تمرین ۴

برای این کار از قدرت نویز ۰.۰۱ شروع می کنیم و تا قدرت نویز ۱.۵ ادامه می دهیم برای هر قدرت نویز ۵۰ بار مقادیر فاصله و سرعت را از سیگنال نویز دار استخراج می کنیم (همانند بخش قبل با این تفاوت که برای این کار یک تابع تعریف شد) و مقادیر خطای سرعت و فاصله را برای آن قدرت نویز میانگین گیری می کنیم در نهایت نمودار میزان خطا بر اساس قدرت نویز برای سرعت به شکل زیر است:



همانطور که مشاهده می گردد تا قدرت نویز ۱.۲ دقت تشخیص سرعت مناسب و قابل قبول است اما پس از آن به طرز سعودی میزان خطا برای تشخیص سرعت افزایش می یابد

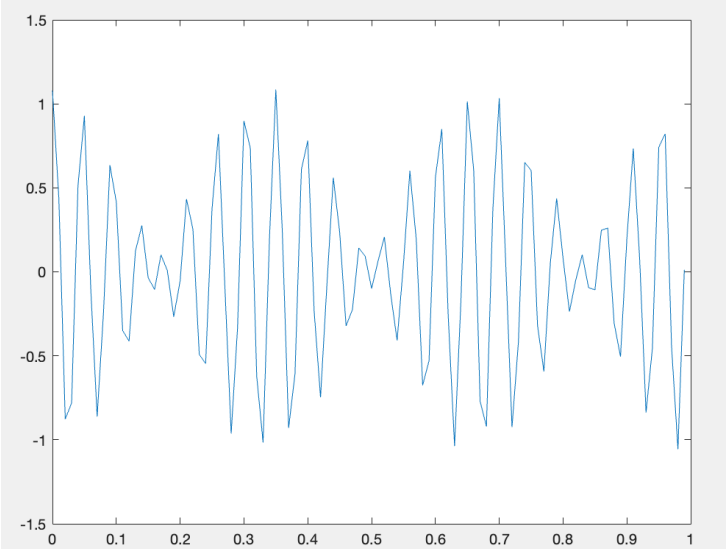
همچنین نمودار خطا بر اساس قدرت نویز برای فاصله به صورت زیر است:



همانطور که مشاهده می شود تشخیص فاصله به شدت نسبت به نویز حساس است و حتی با قدرت نویز های کم نیز دچار خطا می شود. بنابراین می توان نتیجه گرفت که پیدا کردن مقدار فاصله حساسیت بیشتری نسبت به نویز دارد.

تمرین ۵

طبق خواسته ی دستور کار سیگنال رسم می شود:




```
t_start = 0;
t_end = 1;
fc = 5;
fs = 100;
V1 = 180;
R1 = 250;
A1 = 0.5;
V2 = 216;
R2 = 200;
A2 = 0.6;
B = 0.3;
C = 3e8;
p = 2/C;

V1 = V1 / 3.6;
R1 = R1 * 1000;
V2 = V2 / 3.6;
R2 = R2 * 1000;

ts = 1/fs;
t = t_start:ts:t_end-ts;
signal1 = A1*cos(2*pi*(fc+B*V1)*(t-p*R1));
signal2 = A2*cos(2*pi*(fc+B*V2)*(t-p*R2));
signal = signal1+signal2;

plot(t,signal)
```

تمرین ۶

برای این بخش نیز دقیقا همانند بخش ۳ عمل می کنیم با این تفاوت که در تبدیل فوریه ی سیگنال دریافتی دو نقطه ی پیک وجود دارد که در واقعا هر کدام نشان دهنده ی یکی از اجسام هستند باقی مراحل دقیقا همانند بخش سه می باشد برای هر پیک یکبار آن مراحل را انجام می دهیم در نهایت نتیجه به صورت زیر است که کاملا مطابق انتظار است:

```
V1 =

   180

R1 =

  250.0000
```

```
V2 =

   216

R2 =

  200.0000
```

تمرین ۷

از آنجا که سرعت اجسام در واقع تشکیل دهنده ی فرکانس سیگنال دریافتی می باشد (طبق روابط نوشته شده در بخش سوم صورت پروژه و بخش سوم گزارش.) اگر هر دو جسم سرعت یکسان داشته باشند در تبدیل فوریه فقط یک فرکانس غالب (پیک) به دست می آوریم و این موضوع عملا محاسبه فاصله برای هر یک از آن ها غیر ممکن می کند (فقط یک پیک داریم و دو جسم داریم که نیاز به دانستن فاصله ی آن ها داریم !) برای اینکه بتوانیم پارامتر های دو جسم را به درستی تخمین بزنیم باید در تبدیل فوریه دو پیک داشته باشیم این موضوع که این دو پیک رو یکدیگر نیافتند مربوط به رزولوشن فرکانس است که در ابتدای صورت پروژه در رابطه با آن بحث شد در این سوال مقدار رزولوشن فرکانس ۱ است بنابراین دو پیک که وابسته به سرعت هستند باید حداقل به اندازه ی یک واحد از هم فاصله داشته باشند طبق رابطه ی $|fd1 - fd2| = \beta|V1 - V2|$ باید حداقل اختلاف سرعتی را بیابیم که مقدار دو طرف را یک عدد صحیح کند که در این جا با توجه به مقدار $\beta = 0.3$ این مقدار حداقلی برابر ۱۰ متر بر ثانیه است یعنی اختلاف سرعت دو جسم باید حداقل ۱۰ متر بر ثانیه باشد

تمرین ۸

فاصله ی اجسام در فاز سیگنال ورودی تاثیر گذار است بنابراین اثری بر پیدا کردن پیک ها در تبدیل فوریه ندارد بنابراین به آسانی (به شرط یکسان نبودن سرعت ها) می توان ابتدا پیک ها را پیدا کنیم و سپس سرعت اجسام را از روی آن ها بیابیم و پس از آن با استفاده از فاز یکسان فاصله ی هر یک از اجسام (که برابر است) را نیز بیابیم مهم ترین نکته در این جا آن است که یکسان بودن فاصله ی اجسام تاثیر بر تعداد پیک ها در تبدیل فوریه و روی یکدیگر افتادن آن ها ندارند.

تمرین ۹

شرط اولیه این کار آن است که همانطور که در بخش ۷ توضیح داده شد چند جسم سرعت برابر نداشته باشند و حداقل اختلاف سرعت مورد نیاز نسبت به یکدیگر را نیز داشته باشند سپس می توان تعداد آن ها را بر اساس تعداد پیک های تبدیل فوریه ی سیگنال دریافتی پیدا کرد (به تعداد پیک ها جسم داریم !) در ادامه ی کار همانند بخش ۶ به ازای هر یک از پیک ها مقادیر فاصله و سرعت را میابیم.

بخش دوم

تمرین ۱

برای انجام این کار ابتدا باید هر فرکانس را به یک کاراکتر (کاراکتر نت موسیقی) مربوط کنیم تا یک دیتاست تشکیل شود.

```
Mapset = cell(2,12);
Mapset{1,1} = "A";
Mapset{1,2} = "A#";
Mapset{1,3} = "B";
Mapset{1,4} = "C";
Mapset{1,5} = "C#";
Mapset{1,6} = "D";
Mapset{1,7} = "D#";
Mapset{1,8} = "E";
Mapset{1,9} = "F";
Mapset{1,10} = "F#";
Mapset{1,11} = "G";
Mapset{1,12} = "G#";
Mapset{2,1} = 880;
Mapset{2,2} = 932.33;
Mapset{2,3} = 987.77;
Mapset{2,4} = 523.25;
Mapset{2,5} = 554.37;
Mapset{2,6} = 587.33;
Mapset{2,7} = 622.25;
Mapset{2,8} = 659.25;
Mapset{2,9} = 698.46;
Mapset{2,10} = 739.99;
Mapset{2,11} = 783.99;
Mapset{2,12} = 830.61;

save("MAPSET","Mapset");
```

سپس نت ها را به صورت یک رشته متن وارد کرده و مدت زمان نگه داشتن هر نت را نیز در یک ماتریس وارد می کنیم.

```
st = 'DDGF#D|DEEDF#DE|DEF#E|DEEDF#DE|DEDF#E|DEDF#E|DDEF#EF#|F#EF#F#D|' ;
coefficients =[0.5 0.5 1 1 1, 0.5 0.5 0.5 0.5 0.5 0.5 1, 1 1 1 1, 0.5 0.5 (
```

سپس تابعی طراحی می کنیم که رشته متن را گرفته هر نت را در آن شناسایی کرده (بعضی از نت ها دو کاراکتری و بعضی تک کاراکتری هستند بنابراین هر کاراکتر از رشته متن یک نت نیست !) سپس به ازای هر نت باید فرکانس مربوط به آن نت را از دیتا ست بیابیم برای این منظور تابع دیگری طراحی شده که کاراکتر نت را گرفته و فرکانس مربوط را بر می گرداند پس از یافتن فرکانس مناسب هر نت از رشته متن باید سیگنال سینوسی آن فرکانس را با مشخصات داده شده در صورت پروژه بسازیم برای این کار یک تابع دیگر طراحی

شده که فرکانس را به همراه مدت نگه داشتن کلید دریافت می کند و سیگنال سینوسی را بر می گرداند بنابراین تابع اصلی ساخت موسیقی به ازای هر نت با استفاده از دو تابع بالا یک سیگنال سینوسی تولید می کند و این سیگنال ها را با هم جمع کرده و به عنوان سیگنال نهایی موسیقی بر می گرداند.

تابع ساخت سیگنال موسیقی:

```
function signal = create_music(st,coefficients)

st = char(st);
mtx = [];
while ~isempty(st)
    temp = st(1);
    if temp == '|'
        st = st(length(temp)+1:end);
        continue
    end
    if length(st) > 1
        if st(2) == '#'
        temp = [temp '#'];
        end
    end
    mtx = [mtx;string(temp)];
    st = st(length(temp)+1:end);
end
signal = [];
for s=1:length(mtx')
    signal = [signal create_signal(find_freqs(mtx(s,1)),coefficients(s))];
end
end
```

تابع ساخت سیگنال سینوسی:

```
function tone = create_signal(frequency,coefficient)
    fs = 8000;
    ts = 1/fs;
    T = 0.5*coefficient;
    t = 0:ts:T-ts;
    tone = sin(2*pi*frequency*t);
    t_rest = 0:ts:0.025-ts;
    tone = [tone zeros(1,length(t_rest))];
end
```

تابع یافتن فرکانس با توجه به کاراکتر نت:

```
function frequency = find_freqs(note)
    load("MAPSET.mat","Mapset");
    idx = [Mapset{1,:}]==note;
    frequency = Mapset{2,idx};
end
```

حال مدت نگه داشتن هر کلید را به همراه نت ها به تابع ساخت موسیقی می دهیم و نتیجه را می شنویم !

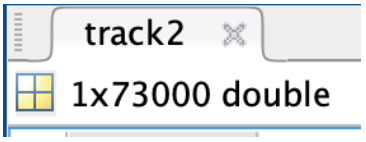
```
track1 = create_music(st,coefficients);
sound(track1, 8000);
```

تمرین ۲

با استفاده از قطعه کد زیر نت موجود در تصویر را تبدیل به موسیقی می کنیم زمان نگه داشتن هر نت نیز در یک ماتریس قابل مشاهده است:

```
st = 'CBAA#C#|CBAA#F|A#FCC#B|BC#CFA#|C#CFA#B|' ;
coefficients = [0.5 1 0.5 0.5 1,0.5 1 0.5 0.5 1,0.5 0.5 0.5 0.5 1,1 0.5 0.5 0.5 1,1 0.5 0.5 0.5 1];

track2 = create_music(st,coefficients);
sound(track2, 8000);

audiowrite('mysong.wav',track2,8000);
```

همچنین مشاهد می شود که موسقی تولید شده داری 73,000 سمپل است و فایل موسیقی تولید شده حجمی برابر با 146,044 بایت دارد یعنی به طور میانگین هر سمپل با ۱۶ بیت ذخیره شده است.



تمرین ۳

برای این کار تابعی طراحی می کنیم که ۱- ابتدا از سیگنال اصلی دریافتی سیگنال مربوط به هر کلید را (که یک سیگنال سینوسی تک تن است.) استخراج کند (در این مرحله مدت زمان نگه داشتن هر کلید نیز مشخص می شود) ۲- سپس از هر سیگنال سینوسی استخراج شده تبدیل فوریه بگیرد و فرکانس اصلی را بیابد و بر اساس فرکانس اصلی و دیتاست داده شده نت مربوط به آن فرکانس را بیابد و نت ها را در یک رشته متن کنار هم بگذارد و بر گرداند. برای بخش اول این تابع یک تابع دیگر طراحی می کنیم که سیگنال اصلی را می گیرد و به دنبال صفر های وسط سیگنال می گردد این صفر ها در واقع همان استراحت های بین فشردن تو کلید هستند پس بین هر دو استراحت یک سیگنال سینوسی تک تن وجود دارد این تابع تک تک این سیگنال ها را استخراج کرده و در یک آرایه ذخیره می کند بر اساس فرکانس نمونه برداری نیز می توان تشخیص داد طول هر سیگنال سینوسی چقدر است و بنابراین بر هر سیگنال تک تن مدت زمان فشردن آن نیز مشخص می شود. و این تابع دو آرایه که یکی شامل سیگنال های تک تن و دیگر مدت زمان فشردن هر کلید است را بر می گرداند برای بخش بخش دوم نیز تابع طراحی می کنیم که همانند بخش های قبلی همین پروژه از سیگنال تک تن سینوسی تبدیل فوریه بگیرد و فرکانس اصلی آن را بیابد سپس این فرکانس اصلی را در دیتاست جستجو کند و نت مربوط به آن را برگرداند. طراحی تابع اصلی به این صورت است:

```
function [notes,durations] = create_notes(signal,fs)

    [temp,durations] = extract_signals(signal,fs);

    notes = [];
    for i = 1:length(temp)
        notes = [notes find_note(temp{i},fs,durations(i))];
    end
    notes = string(notes);

end
```

تابع استخراج سیگنال های تک تن از سیگنال اصلی:

```
function [tones,durations] = extract_signals(signal,fs)
    tones = {};
    durations = [];
    ts = 1/fs;

    while ~isempty(signal)
        first = find(signal);
        if isempty(first)
            break;
        end
        first = first(1)-1;
        last = first;
        if last == length(signal)-5
            break;
        end

        while any(signal(last:last+5))
            last = last + 1;
        end
        last = last - 1;
        tones{end+1} = [signal(first:last)];
        signal = signal(last+1:end);
        durations = [durations ts*(last-first++1)];
    end
end
```

تابع یافتن نت بر اساس فرکانس سیگنال تک تن که با استفاده از تبدیل فوریه به دست آمده:

```
function note = find_note(signal,fs,T)

    load("MAPSET.mat","Mapset");
    resol = 1/T;
    freq = -fs/2:resol:fs/2-resol;
    FT = abs(fftshift(fft(signal)));

    FT = FT(freq>=0);
    freq = freq(freq>=0);
    [~,fc] = max(FT);
    fc = freq(fc);

    tones = [Mapset{2,:}];
    tones = abs(tones - fc);
    [~,idx] = min(tones);
    note = char(Mapset{1,idx});

end
```

در نهایت تابع نوشته شده را برای تمرین اول این بخش تست می کنیم:

```
[decoded_notes, decoded_coefficients] = create_notes(track1,8000);


st1
decoded_notes
coefficients1
decoded_coefficients
```

نتایج به دست آمده کامل صحیح و منطبق هستند:

نت ها:

```
st1 =

    'DDGF#D|DEEDF#DE|DEF#E|DEEDF#DE|DEDF#E|DEDF#E|DDEF#EF#|F#EF#F#D|'

decoded_notes =

    "DDGF#DDEEDF#DEDEF#EDEEDF#DEDEDF#EDEDF#EDDEF#EF#F#EF#F#D"
```

زمان نگه داشتن هر نت:

```
  coefficients1 =
    Columns 1 through 10
      0.5000    0.5000    1.0000    1.0000    1.0000    0.5000    0.5000    0.5000    0.5000    0.5000
    Columns 11 through 20
      0.5000    1.0000    1.0000    1.0000    1.0000    1.0000    0.5000    0.5000    0.5000    0.5000
    Columns 21 through 30
      0.5000    0.5000    1.0000    1.0000    0.5000    0.5000    1.0000    1.0000    1.0000    0.5000
    Columns 31 through 40
      0.5000    1.0000    1.0000    0.5000    0.5000    1.0000    0.5000    0.5000    1.0000    0.5000
    Columns 41 through 44
      0.5000    1.0000    1.0000    1.0000


decoded_coefficients =
  Columns 1 through 10
    0.5000    0.5000    1.0000    1.0000    1.0000    0.5000    0.5000    0.5000    0.5000    0.5000
  Columns 11 through 20
    0.5000    1.0000    1.0000    1.0000    1.0000    1.0000    0.5000    0.5000    0.5000    0.5000
  Columns 21 through 30
    0.5000    0.5000    1.0000    1.0000    0.5000    0.5000    1.0000    1.0000    1.0000    0.5000
  Columns 31 through 40
    0.5000    1.0000    1.0000    0.5000    0.5000    1.0000    0.5000    0.5000    1.0000    0.5000
  Columns 41 through 44
    0.5000    1.0000    1.0000    1.0000
```