باسمه تعالی

درس سیگنال‌ها و سیستم‌ها

گزارش تمرین متلب

سری اول

امیرحسین افشارراد

۹۵۱۰۱۰۷۷

بهراد منیــری

۹۵۱۰۹۵۶۴

۱۳۹۶ / ۱۲ / ۴

سؤال اول) پردازش سیگنال صوت

۱ـ ۱)

تابع SoundMaker، سه بردار gain، note و interval و fs را به عنوان ورودی گرفته و سیگنال صوتی متناظر را به عنوان خروجی برمی‌گرداند.

```
function [ Y ] = SoundMaker( gain, note, interval, Fs)
    a = length(gain)
    Y = 0;
    for i = 1 : a
        f = 440.*2.^(note(i)./12);
        omega = 2 .* pi .* f;
        t = 0 : 1/Fs : interval(i);
        Y = [Y , gain(i) .* sin(omega.*t)];
    end
end
```

این تابع، سیگنال های سینوسی با فرکانس مربوط به هر note و با دامنه و طول داده‌شده را پشت سر هم قرار می‌دهد.

۱ـ ۲،۳،۴)

با تابعی که در بخش قبل نوشته شده، سیگنال صوتی خواسته شده، m1، را می‌سازیم. از تابع sound برای پخش صدا استفاده کرده و با کمک تابع audiowrite، فایل صوتی مربوط به سیگنال m1 را تولید می‌کنیم.

```
load('G_N_I.mat');
Fs = 200000;
m1 = SoundMaker(gain, note, interval, Fs);
sound(m1, Fs);
filename = 'music.wav';
audiowrite(filename, m1, Fs);
```

۱ـ ۵)

F_s را به مرور کاهش می‌دهیم، بعد از $F_s \approx 80\ kHz$ صدا دیگر به خوبی شنیده نمی‌شود.

۱ـ ۶)

تعداد نقاط سیگنال ثابت است زیرا توسط تابع SoundMaker تولید شده و در آن تغییری داده نمی‌شود. اگر فرکانس نمونه‌گیری را کاهش دهیم، زمان موسیقی افزایش می‌یابد. زیرا فاصلهٔ زمانی بین نمونه ها زیاد می‌شود.

$F_s \approx 20\ kHz$: در این فرکانس تقریبا صدای مفیدی شنیده نمی‌شود.

$F_s \approx 100\ kHz$: در پخش با این فرکانس، کیفیت صوت آسیب زیادی نمی‌بیند و تنها طول هر نت به همان دلیلی که در بالا ذکر شد افزایش می‌یابد.

$F_s \approx 400\ kHz$: فرکانس نمونه‌گیری مورد قبول تابع sound باید در بازهٔ $1000 < F_s < 384000$ هرتز باشد. به همین دلیل تابع sound در این فرکانس نمونه‌گیری خطا می‌دهد.

۷ـ۱)

افزودن مقدار DC به interval باعث می‌شود هر نت به مدت طولانی تری نواخته شود و افزودن مقدار DC به note باعث می‌شود نت ها به طور یکنواخت زیر تر یا بم تر بشوند.
این موضوع دقیقا با تعریف هر بردار نیز هم‌خوانی دارد.

۸ـ۱)

1. افزودن نویز به interval
این کار باعث می‌شود که در زمان اجرای هر نت ناهمگونی به وجود بیاید، برخی نت ها طولانی تر از حد معمول و برخی نت ها کوتاه‌تر از حد معمول شنیده می‌شوند.

2. افزودن نویز به notes
با این کار در نت های اشکال ایجاد می‌شود و فرکانس برخی کم و برخی زیاد می‌شود. از آنجایی که میانگین نویز صفر است تا حدودی شکل اولیه آهنگ حفظ می‌شود اما بعضا برخی نت ها به شدت آسیب می‌بینند.

3. افزودن نویز به gains
افزودن نویز به gain باعث تغییر در دامنه سینوسی ها می‌شود. به نظر می‌رسد گوش به تغییر دامنه سینوسی‌ها کمتر حساس است و بیشتر به فرکانس ها توجه می‌کند و با این کار آسیب چندانی به موسیقی نمی‌رسد.

به نظر تاثیر نویز بر روی interval به دلیل آسیب زدن به نت ها تاثیر کلی مخرب‌تری دارد زیرا، از نظر من، پیوستگی موسیقی عامل اصلی زیبایی آن است و افزودن نویز به طول بازه های زمانی این پیوستگی را از بین می‌برد.

۹ـ۱)

در این بخش یک نویز با میانگین صفر و واریانس یک را به سیگنال اضافه می‌کنیم. یک صدای خش خش به صوت افزوده می‌شود.

```
s = size(m1);
noise = (randn(s(2),1))'; % Gaussian noise with sigma = 1 and mean = 0
noisySignal = m1 + noise;
```

۱۰ـ۱)

در این بخش به سیگنال اصلی `a.*(randn(s(2),1))'` را اضافه می‌کنیم. از درس می‌دانیم توان سیگنال به صورت زیر تعریف می‌شود.

$$P = \frac{Energy}{N+1} = \frac{\sum_{n=0}^{N} x^2[n]}{N+1}$$

فرض کنید توان نویز گاوسی با واریانس یک برابر P_0 باشد در نتیجه نویز جدید که در بالا معرفی شد توانی برابر a^2P_0 است.
پس با کنترل a میتوان انرژی نویز را تغییر داد. به مرور a را افزایش می‌دهیم. هنگامی که انرژی نویز $P_1 = 400$ می‌شود، صدای موسیقی تقریبا غیر قابل شنیدن می‌شود. مقدار a متناظر با این توان 20 است.

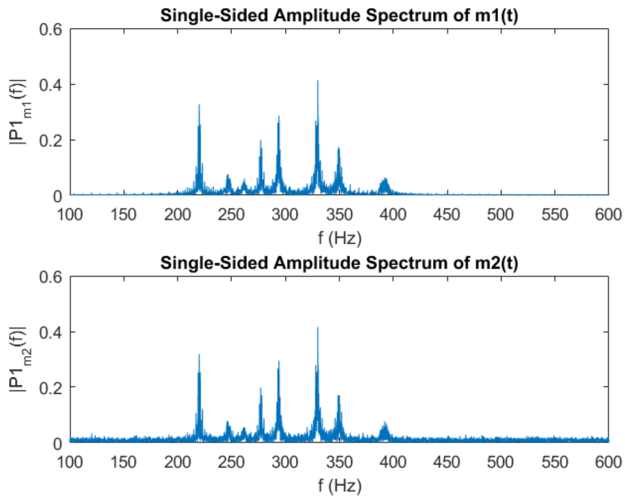
(۱ــ۱۱)

از رابطه توان واضح است برای یک پنجم کردن توان نویز نسبت به بخش دهم، باید از $a = \frac{a_0}{\sqrt{5}} = \frac{20}{\sqrt{5}}$ استفاده کنیم.

```
s = size(m1);
noise = 20./sqrt(5) .*(randn(s(2),1))';
m2 = m1 + noise;
sound(m2, Fs);
newPower = sum(noise.*noise)./s(2)
filename = 'noisyMusic.wav';
audiowrite(filename, m2, Fs);
```

این صوت در فایل noisyMusic.wav ذخیره شده و در ضمیمهٔ گزارش آمده است.

(۱ــ۱۲)



شکل ۱

سیگنال ما یک سیگنال گسسته در زمان است. برای این سیگنال Discrete Time Fourier Transform را با کمک تابع fft محاسبه می‌کنیم. به دلیل حقیقی بودن سیگنال، تبدیل فوریه آن نسبت به محور عمودی قرینه است پس توجه خود را تنها به فرکانس‌های مثبت معطوف می‌کنیم و می‌دانیم که در فرکانس‌های منفی نیز شکل تبدیل فوریه همین است.

نمودار اول تبدیل فوریه صوت بدون نویز و نمودار دوم تبدیل فوریه صوت و نویز است. دیده می‌شود که نویز گاوسی در تمام فرکانس ها محتوای فرکانسی دارد و مقدار اندکی به تمام فرکانس ها اضافه شده است.

در بخش‌های بعدی از همین موضوع برای فیلتر کردن نویز استفاده خواهیم کرد.

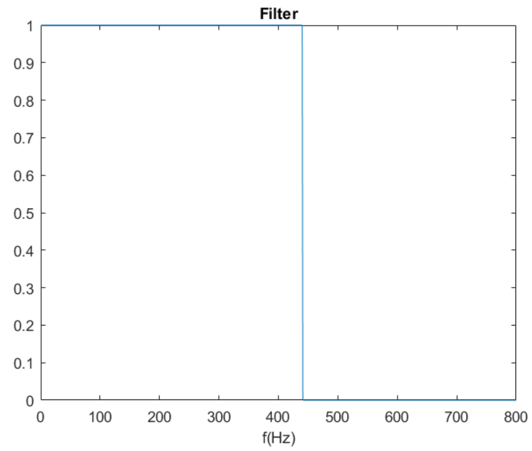
تابع plotFFT، تابعی برای رسم نمودار تبدیل فوریه سیگنال حقیقی به فرمت نمودار های بالاست که محور افقی آن بر حسب فرکانس است.

۱۳ــ۱)

سیگنال m1 در حوزهٔ زمان یک تابع قطعه‌ای سینوسی است که هر سینوسی فرکانسی مربوط به نت خود را دارد.
از رابطهٔ فرکانس هر نت در می‌یابیم که فرکانس سینوسی ها در بازهٔ $220 < f < 440 \times 2^{-15}$ هرتز است.
اگر توجه خود را تنها به فرکانس های مثبت معطوف کنیم، تبدیل فوریه سینوسی با فرکانس ω ضربه‌ای است در فرکانس ω. در این سوال سیگنال های سینوسی کامل نداریم و قطعه های سینوسی داریم و انتظار داریم که در فرکانس هایی که سینوسی با آن فرکانس داریم، تبدیل فوریه مقدار زیادی داشته باشد که این اتفاق در نمودار های فوق رخ داده است.

۱۴ــ۱)

انتظار داریم که صوت بدون نویز شامل فرکانس های بیش از ۴۴۰ هرتز نباشد پس می‌توانیم تمام فرکانس های بالای ۴۴۰ هرتز را صفر کنیم.
برای این کار باید سیگنال را در حوزهٔ فرکانس در یک rect ضرب شود که معادل کانولوشن زمانی با sinc است.
برای این کار تبدیل فوریه را در تابع زیر، که یک فیلتر پایین گذر است ضرب می‌کنیم.



شکل ۲

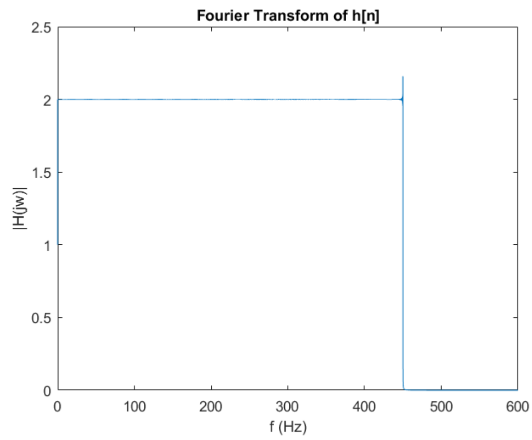
اگر از فرکانس قطع $F_c = 450\ Hz$ استفاده کنیم می‌توانیم تمام موسیقی ها را فیلتر کنیم چون فرکانس های ممکن در موسیقی ما کمتر از این مقدار است. این کار بخشی از نویز با فرکانس بالا را حذف می‌کند اما نویز با فرکانس پایین باقی می‌ماند. با ساختن یک فیلتر میان‌گذر می‌توان این بخش از نویز را نیز فیلتر کرد ولی چون نویز گاوسی در تمام فرکانس ها، حتی فرکانس های موسیقی، نیز حضور دارد، بدون آسیب به اصل موسیقی کل نویز را نمی‌توان حذف کرد.

۱ـ۱۵)

در این بخش تلاش می‌کنیم تا تابع sinc متناظر را فیلتر پایین گذر با فرکانس قطع 450 Hz را ساخته و به کمک آن نویز را فیلتر کنیم. تابع زیر را در نظر بگیرید.

$$h[n] = 2\frac{450}{F_s}sinc(2 \times \frac{450}{F_s}n)$$

تبدیل فوریه این تابع به صورت زیر است.



شکل ۳

با دیدن نمودار تبدیل فوریه فیلتر می‌بینیم که پدیدهٔ گیبس منجر به ایده‌آل نبودن فیلتر شده است و پالس مقداری overshoot نیز دارد که در بخش بعد به کمک روش همینگ آن را اصلاح می‌کنیم.

با این وجود فیلتر فوق با تقریب خوبی فیلتر مدنظر ماست بنابراین صوت را با آن فیلتر می‌کنیم.

$$x_{filtered} = h[n] * m_2[n]$$

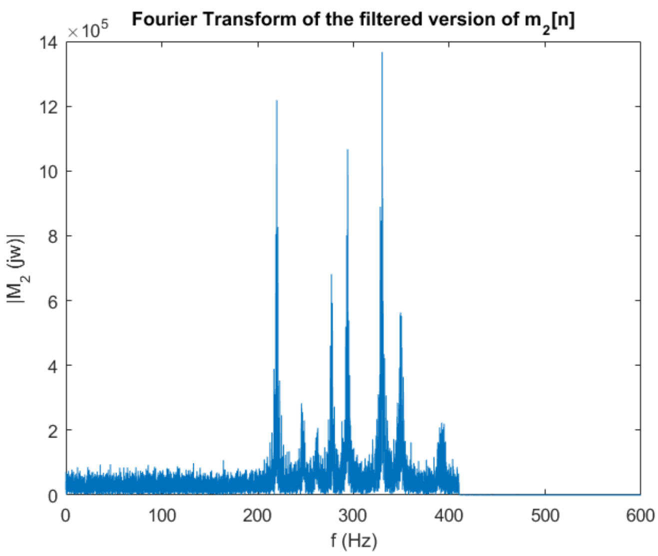
با استفاده از تابع conv متلب کانولوشن گسسته فوق را محاسبه کردیم. نتیجه این کار در فایل m2_f.wav به ضمیمه آمده است اما این کار به حدود یک ساعت محاسبه نیاز دارد؛ روش سریع تر انجام محاسبات در حوزه فرکانس است:

$$x_{filtered} = h[n] * m_2[n] \rightarrow X_{filtered} = H(jw)M_2(jw)$$

و سپس از $X_{filtered}$ تبدیل فوریه وارون می‌گیریم تا $x_{filtered}[n]$ به دست بیاید.

در فایل main.m صوت با هر دو روش کانولوشن زمانی و ضرب در حوزهٔ فرکانس فیلتر شده است.

تبدیل فوریه نتیجه به این صورت است:



شکل ٤

۱۶ــ۱)

فیلتر بخش قبل دو ایراد عمده داشت:

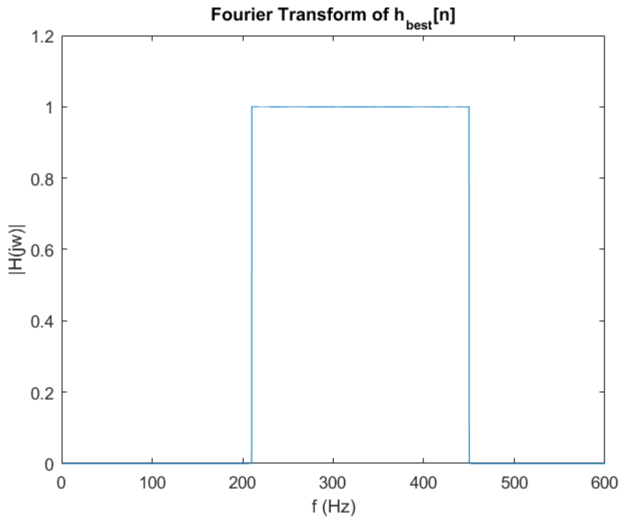
۱ ــ باقی ماندن نویز فرکانس پایین بعد اعمال فیلتر

۲ ــ پدیده Gibbs و وجود overshoot در حوزهٔ فرکانس

برای رفع مشکل اول فیلتر را به جای پایین گذر، پایین گذر می‌سازیم و برای رفع مشکل دوم از روش Hamming استفاده می‌کنیم.
فیلتر میان‌گذرمان را به این صورت می‌سازیم

```
hsupp = (-(L-1)/2:(L-1)/2);
fc = 450;
h1 = (2*fc/Fs)*sinc(2*fc*hsupp/Fs).*hamming(L)';
fc = 210;
h2 = (2*fc/Fs)*sinc(2*fc*hsupp/Fs).*hamming(L)';
h = h1 -h2;
```

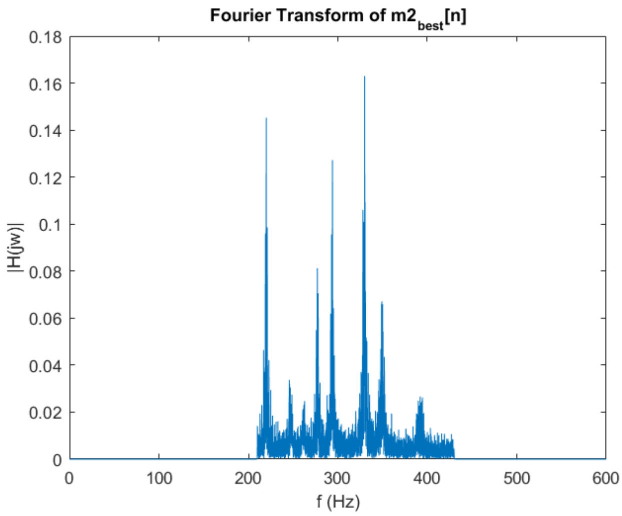
فیلترمان را در hamming ضرب کردیم تا اثرات overshoot به حداقل برسد زیرا این تابع کمک می‌کند تا تابع sinc زودتر به صفر برسد و خطای انرژی سری فوریه زودتر و با جملات کمتری به صفر همگرا شود. تبدیل فوریه فیلتر فوق به صورت زیر است.



شکل ٥

با اعمال این فیلتر به سیگنالمان می‌توان محتوای نویز را در تمام فرکانس ها به جز ۲۲۰ تا ۴۵۰ را کاملا حذف کنیم. صوت حاصل در فایل m2_best.wav ضمیمه شده است.

نمودار تبدیل فوریه سیگنال نویزی بعد از اعمال فیلتر فوق:



شکل ٦

سؤال دوم) کانولوشن

ابتدا تابع MyConv را تشکیل دادیم. برای این کار، از مفهوم ریاضی ماتریس کانولوشن استفاده نموده‌ایم. ماتریس کانولوشن، روشی برای محاسبه کانولوشن دو بردار است، به شکلی که ابتدا با استفاده از یکی از این دو بردار، یک ماتریس با شرایط مشخصی ساخته می‌شود، سپس از ضرب این ماتریس در بردار دیگر، بردار جدیدی حاصل می‌شود که همان کانولوشن دو بردار اولیه است. اگر دو بردار مورد نظر، $x^T = [x_1, x_2, ..., x_n]$ و $y^T = [y_1, y_2, ..., y_m]$ باشند، ماتریس کانولوشن به شکل زیر ساخته می‌شود: (در این جا، ماتریس کانولوشن را با استفاده از بردار x ساخته‌ایم. مسأله کاملاً متقارن است و می‌توان آن را با استفاده از بردار y نیز ساخت)

$$M = \begin{bmatrix} x_1 & 0 & \cdots & 0 \\ \vdots & x_1 & & \vdots \\ x_n & \vdots & & \vdots \\ 0 & x_n & \ddots & 0 \\ \vdots & 0 & & x_1 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & x_n \end{bmatrix}_{(m+n-1)\times m}$$

حال، حاصل کانولوشن به راحتی از رابطه‌ی $x * y = M \times y$ به دست می‌آید. همچنین ابعاد این مسأله نیز سازگار است. M ماتریسی $(m+n-1)\times m$ و y برداری $m \times 1$ است و حاصل کانولوشن، برداری به طول $m+n-1$ است.

برای تولید ماتریس کانولوشن، می‌توانیم در متلب از تابع **<u>convmtx</u>** استفاده کنیم که با گرفتن بردار x در ورودی، ماتریس کانولوشن متناظر با آن را تولید می‌کند. بنابراین مسأله به سادگی و با کد output = convmtx(x)*y حل می‌شود.

اما برای آن که کمی بیشتر با مسأله درگیر شویم، در کد برنامه، ماتریس کانولوشن را از طریق دیگری ساختیم. برای این کار از تابع **toeplitz** استفاده کردیم. این تابع دو ورودی به شکل toeplitz(c,r) دارد. c و r دو بردار هستند که با استفاده از آن‌ها، یک ماتریس ساخته می‌شود. نحوه عملکرد این تابع به این صورت است که بردار c را در سطر اول و بردار r را در ستون اول ماتریس قرار می‌دهد. (باید درایه اول دو بردار یکسان باشد) در ادامه، سطر و ستون دوم را با شیفت سطر و ستون اول به سمت راست و پایین تولید می‌کند. بدیهی است درایه آخر c و r، در سطر و ستون دوم بیرون از ماتریس می‌افتد و دیگر در سطر و ستون دوم ظاهر نمی‌شوند. به همین ترتیب با شیفت راست و پایین، سطر و ستون‌های بعدی هم تولید می‌شود تا ماتریس کامل شود. (تا زمانی که یکی از دو بردار c یا r تمام شود) نمونه‌ای از کارکرد این تابع به شکل زیر است:

```
c = [1  2  3 4];
r = [4 5 6];
toeplitz(c,r)
```

```
Warning: First element of input column does not match first element of input
row.
         Column wins diagonal conflict.

ans =

     1     5     6
     2     1     5
     3     2     1
     4     3     2
```

شکل ۷

همچنین در شکل ۷، توضیح داده شده که در صورت عدم تطابق درایه اول دو بردار ورودی، بردار ستونی (c) پیروز می‌شود.

حال، باید ببینیم چگونه می‌توان از این تابع برای تولید ماتریس کانولوشن استفاده کرد. این کار، به آسانی قابل انجام است. برای تشکیل بردار c، ابتدا بردار x را قرار داده و سپس در ادامه آن، به اندازه $m-1$ عدد صفر قرار می‌دهیم. (m طول بردار y است)

برای تشکیل بردار r، نیز در ابتدای آن، درایه اول بردار x را قرار داده و سپس، $m-1$ عدد صفر قرار می‌دهیم. با دادن این دو بردار به تابع toeplitz، خروجی، همان ماتریس کانولوشن مورد نظر است.

کد برنامه نیز به شرح زیر است:

```
function Y = MyConv(u,v)
    c = [u zeros(1,length(v)-1)];
    r = [u(1) zeros(1,length(v)-1)];
    ConvolutionMatrix = toeplitz(c,r);
    Y = ConvolutionMatrix*v';
    Y = Y';
end
```

در ادامه، چهار کانولوشن خواسته شده را انجام می‌دهیم. ابتدا، ذکر چند نکته خالی از لطف نیست.

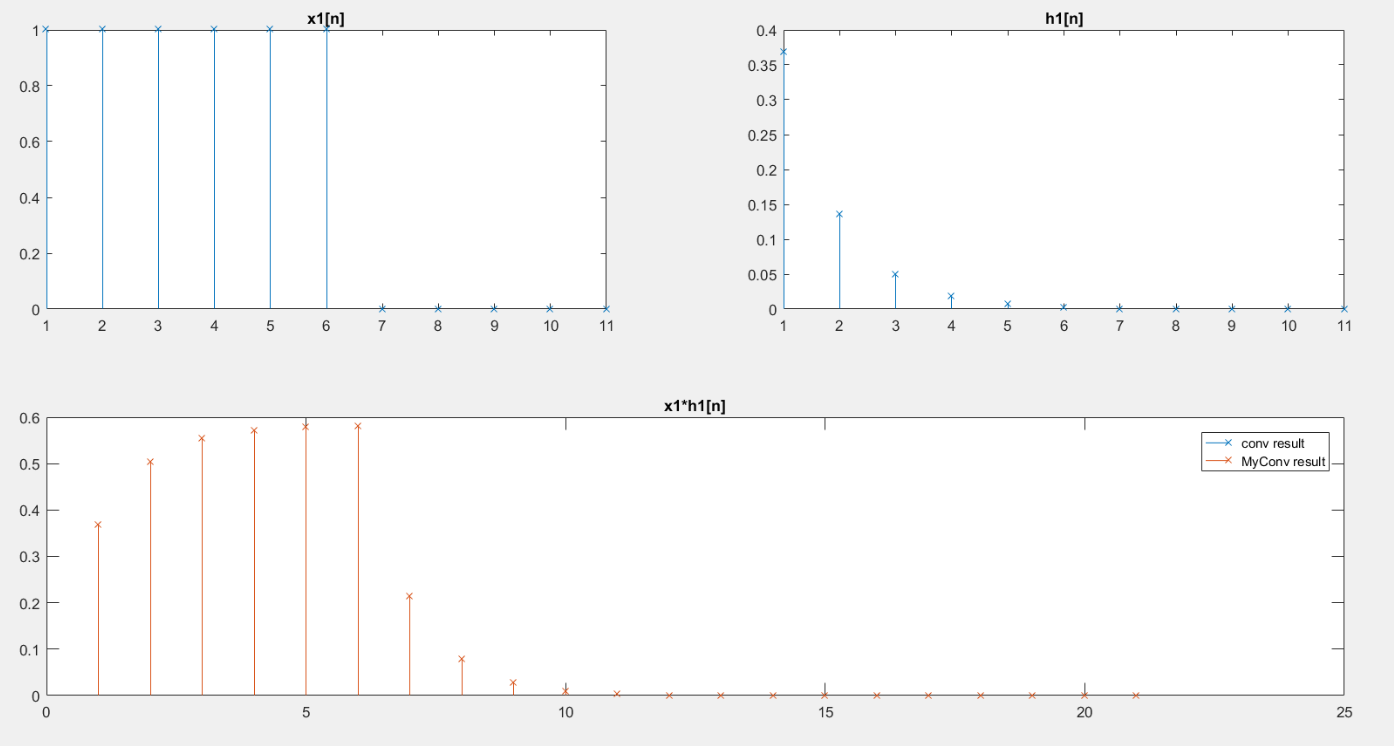
نکته۱) در نمونه‌های گسسته، سیگنال‌ها را مطابق با صورت سؤال تعریف کرده‌ایم و در نمونه‌های پیوسته، با فرکانس Fs مشخص شده در کد برنامه نمونه‌برداری کرده‌ایم. همچنین این نکته نیز رعایت شده است که فرکانس نمونه‌برداری باید بیشتر از دو برابر بزرگ‌ترین فرکانسی باشد که در سیگنال اصلی موجود است.

نکته۲) برای سیگنال‌های پیوسته، شروع نمونه‌برداری را متناظر با $n = 0$ در نظر گرفتیم. در سیگنال‌های گسسته، خود سیگنال مقیاس محور زمان (n) را مشخص کرده است.

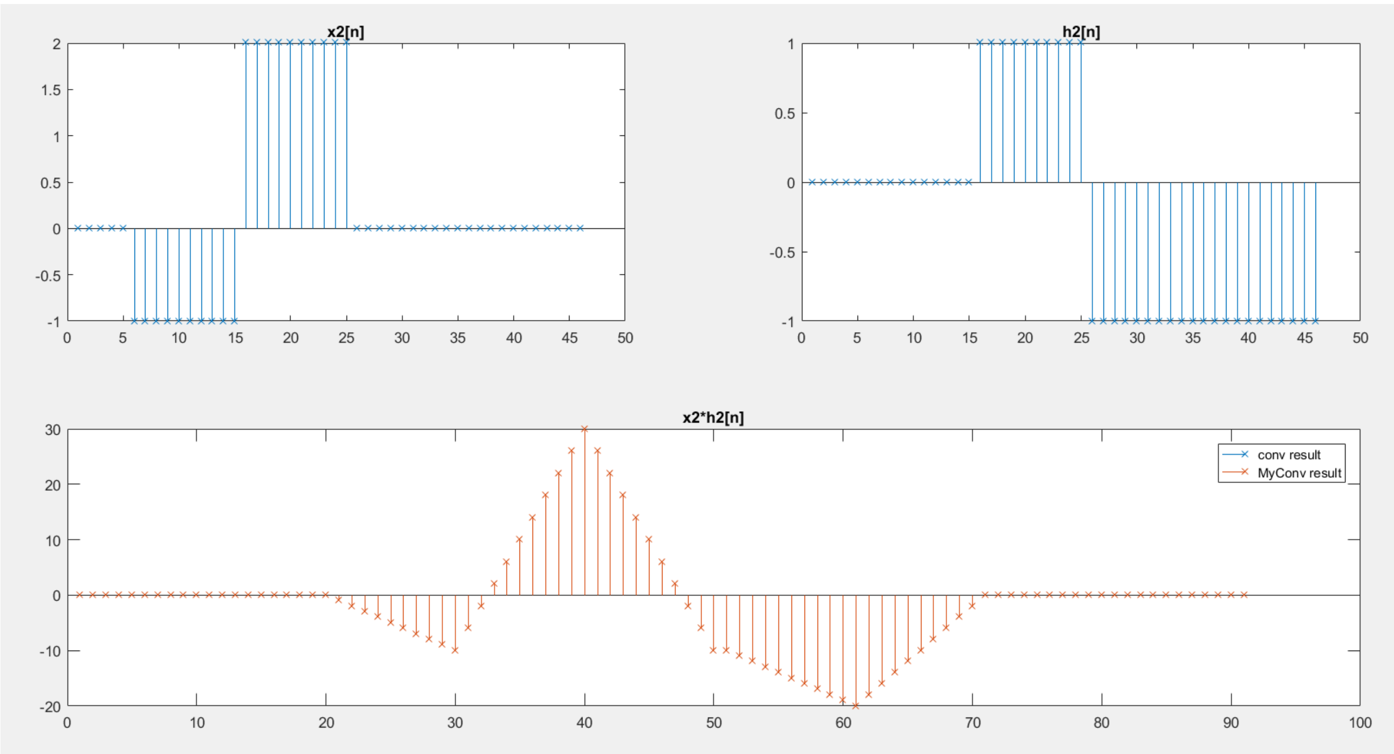
نکته۳) در کانولوشن مورد c، دو تابع سینوسی با فرکانس‌های ۱ و ۱۰۰ هرتز با هم جمع شده‌اند. چنین تابعی، یک سینوسی ۱ هرتز است که یک سینوسی ۱۰۰ هرتز روی آن سوار شده است. با تغییر جزئی فرکانس نمونه‌برداری، ممکن است نمودار x[n] شکل جدید و متفاوتی را پیدا کند که علت آن، تغیرات سریع تابع است، ولی مقادیر نمونه‌ها با فرکانس‌های نمونه‌برداری متفاوت آن قدر نزدیک و سازگار هستند که حاصل کانولوشن، تحت تأثیر فرکانس نمونه‌برداری قرار نمی‌گیرد. (با فرض رعایت قضیه Nyquist)

نکته۴) در تمامی موارد، حاصل کانولوشن با استفاده از توابع conv و MyConv را بر روی یک نمودار رسم کرده‌ایم که یکی آبی و دیگری نارنجی است. (در راهنمای نمودار نیز قابل مشاهده است) که چون این دو نمودار دقیقا منطبق هستند (نشان از عملکرد درست تابع MyConv دارد)، فقط یکی از این دو نمودار قابل مشاهده است.

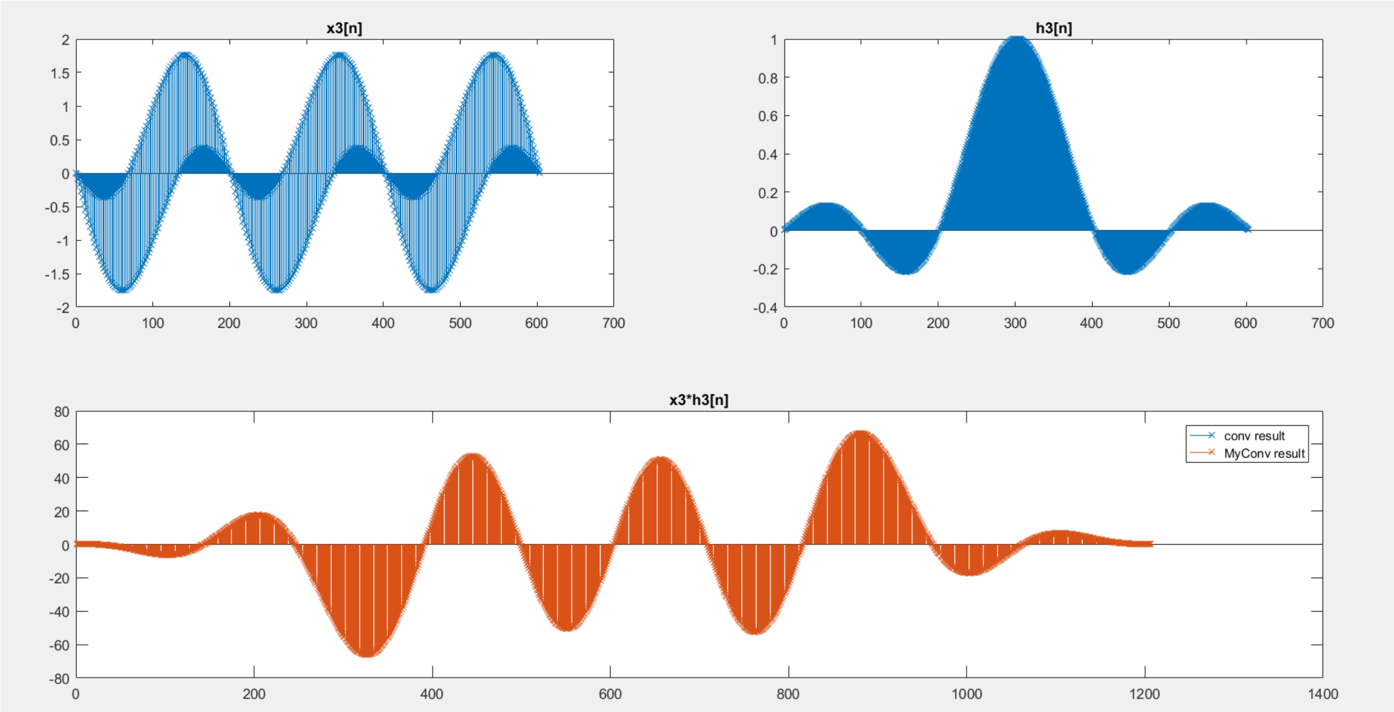
در صفحات بعد، نمودارهای رسم شده برای موارد a و b و c و d را مشاهده می‌کنید.


x1[n]
h1[n]
x1*h1[n]
conv result
MyConv result

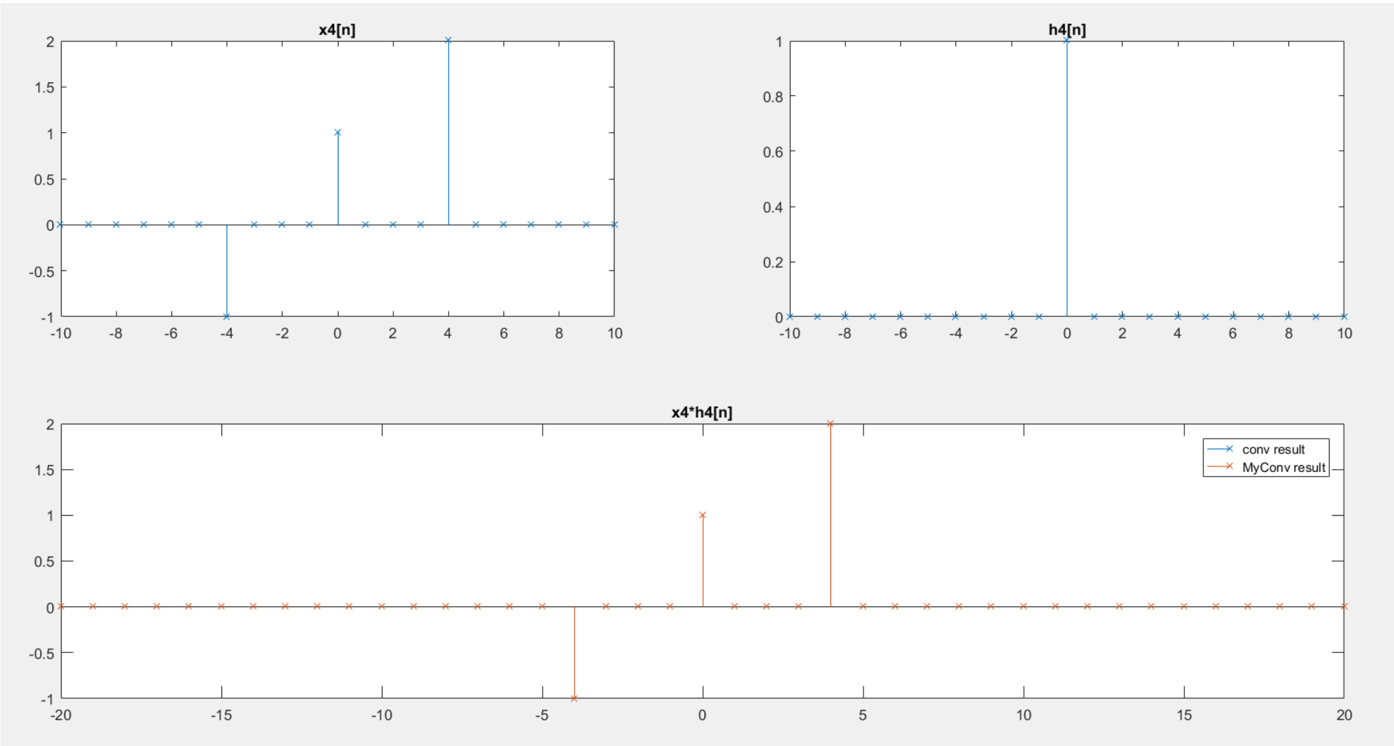
شکل ۸


x2[n]
h2[n]
x2*h2[n]
conv result
MyConv result

شکل ۹


x3[n]
h3[n]
x3*h3[n]
conv result
MyConv result

شکل ۱۰


x4[n]
h4[n]
x4*h4[n]
conv result
MyConv result

شکل ۱۱