به نام خدا

1 . کانولوشن

1.1. محاسبه کانولوشن

1) می دانیم برای سیگنال های پیوسته در زمان، کانولوشن دو سیگنال با استفاده از رابطه زیر محاسبه می شود:

$$x_1(t) * x_2(t) = \int_{-\infty}^{+\infty} x_1(\tau)x_2(t-\tau)\,d\tau$$

همچنین برای سیگنالهای گسسته در زمان داریم :

$$x_1[n] * x_2[n] = \sum_{k=-\infty}^{\infty} x_1[k]x_2[n-k]$$

برای دو سیگنال داده شده در صورت سوال داریم:

$$x_1[n] * x_2[n] = \sum_{k=-\infty}^{\infty} \left(\frac{1}{3}\right)^k (u[k]-u[k-10])(0.9)^{n-k}(u[n-k]-u[n-k-5])$$

$$= \sum_{k=0}^{10} \left(\frac{1}{3}\right)^k (0.9)^{n-k}(u[n-k]-u[n-k-5])$$

$$= (0.9)^n \sum_{k=0}^{10} (2.7)^{-k}(u[n-k]-u[n-k-5])$$

در نتیجه برای $0 \le n < 5$ کانولوشن دو سیگنال به صورت زیر خواهیم بود:

$$(0.9)^n \sum_{k=0}^{n} (2.7)^{-k}$$

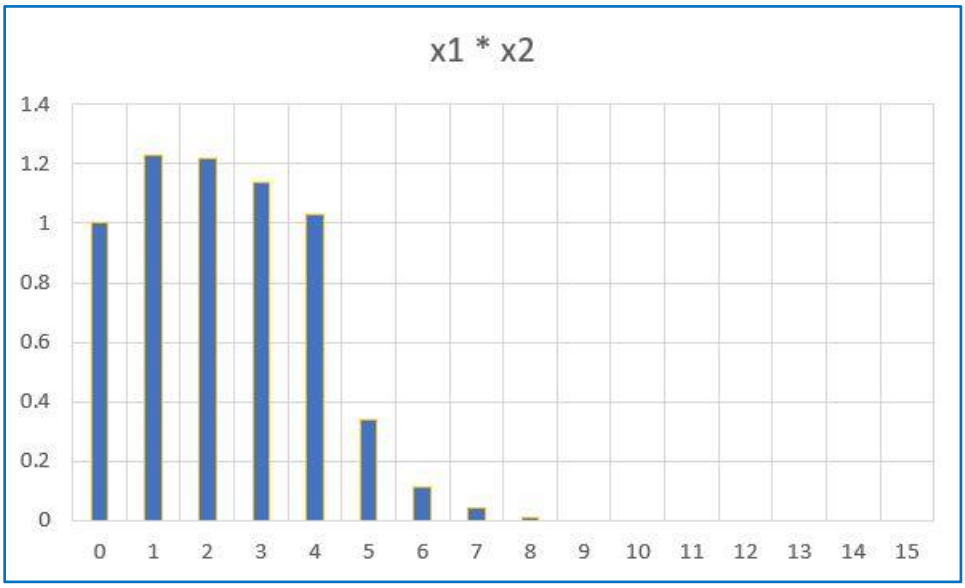
و برای $5 \le n \le 10$ کانولوشن دو سیگنال به صورت زیر خواهد بود:

$$(0.9)^n \sum_{k=n-5}^{n} (2.7)^{-k}$$

و در نهایت برای $10 < n \le 15$ داریم:

$$(0.9)^n \sum_{k=n-5}^{10} (2.7)^{-k}$$

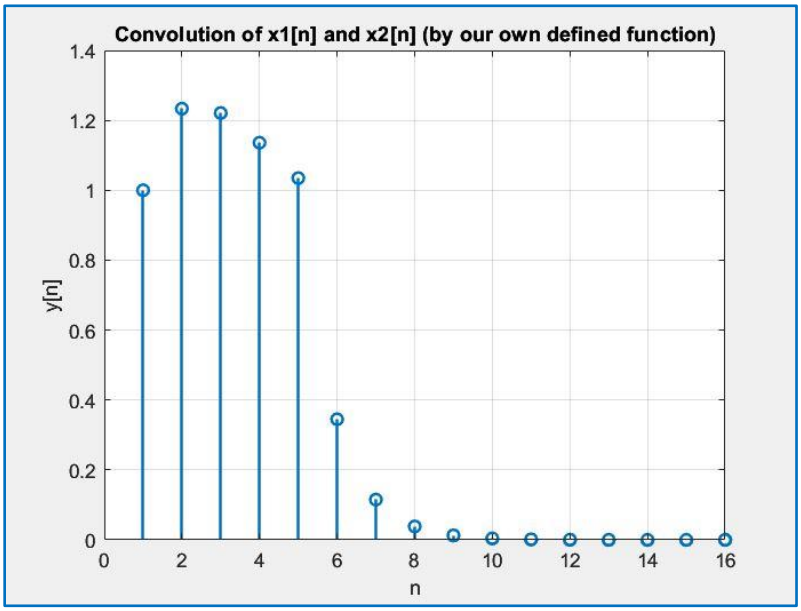
بدیهی است که برای $n > 15$ و $n < 0$ حاصل کانولوشن دو سیگنال صفر خواهد بود.



شکل1 – کانولوشن دو سیگنال داده شده در سوال 1

مقادیر کانولوشن دو سیگنال را به صورت عددی و با استفاده از رابطه های بدست آمده در صفحه قبل بدست آوردیم و در نرم افزار اکسل نمودار آن را رسم کردیم. همانطور که دیده می شود از دو سیگنال به طول های 11 و 6 کانولوشن گرفتیم و طول سیگنال نهایی برابر با 16 شده است. بنابراین می توان گفت کانولوشن دو سیگنال به طول های n و m سیگنالی به طول n+m-1 به ما خواهد داد.

2) حال در نرم افزار متلب تابع my_conv را می نویسیم و با تعریف کردن دو سیگنال داده شده و دادن آن ها به تابع مورد نظر، کانوولوشن آن ها را حساب می کنیم. سیگنال خروجی به صورت زیر خواهد بود:



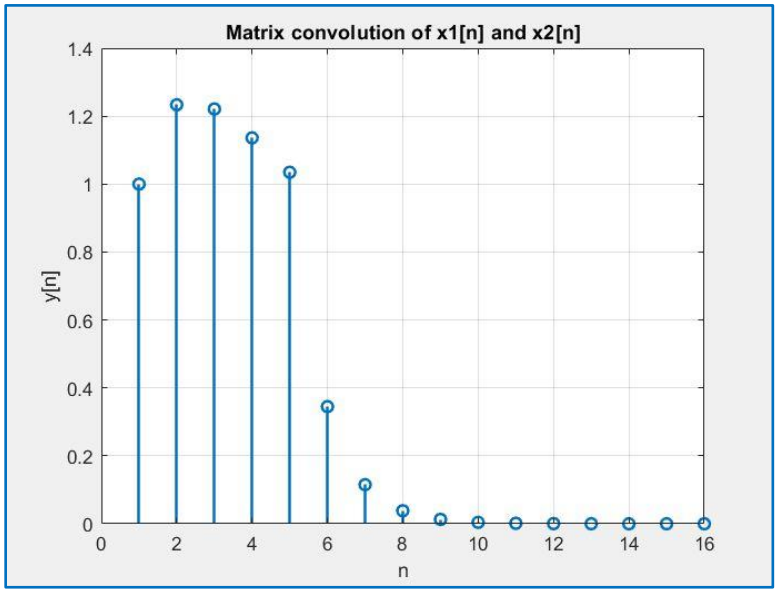
شکل2 – محاسبه کانولوشن دو سیگنال داده شده در سوال 1 با استفاده از تابع نوشته شده توسط خودمان

دقت شود که از آنجایی که اندیس آرایه ها در متلب از 1 شروع می شود، ترسیم نمودار از نقطه n=1 شروع شده و یک شیفت به جلو دارد اما به طور کلی پاسخ منطبق بر جواب بدست آمده در بخش قبل است.

3) در این بخش تابع my_matrix_conv می نویسیم که کانولوشن دو تابع را به صورت ماتریسی به روش گفته شده در صورت سوال محاسبه می کند.

روش در پیش گرفته شده در این تابع به این صورت است که دو سیگنال ورودی که به صورت آرایه گرفته شده اند را به ماتریس های سطری و ستونی تبدیل می کند و در هم ضرب می کند تا یک ماتریس مربعی ایجاد شود؛ سپس در راستای قطر فرعی درایه ها را با هم جمع می زند و سیگنال خروجی را محاسبه می کند.

سیگنال خروجی به صورت زیر خواهد بود:



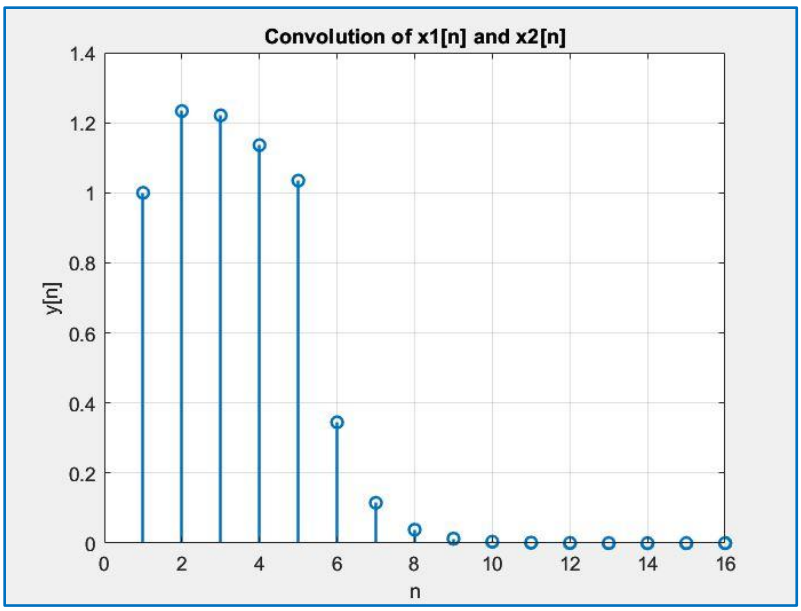
شکل3 – محاسبه کانولوشن دو سیگنال داده شده در سوال 1 با استفاده از روش ماتریسی

دقت شود که از آنجایی که اندیس آرایه ها در متلب از 1 شروع می شود، ترسیم نمودار از نقطه n=1 شروع شده و یک شیفت به جلو دارد اما به طور کلی پاسخ منطبق بر جواب بدست آمده در بخش قبل است.

4) در این بخش با استفاده تابع آماده conv در متلب، کانولوشن دو سیگنال را محاسبه می کنیم.

سیگنال خروجی در صفحه بعد آورده شده است.

همانطور که دیده می شود کانولوشن محاسبه شده در بخش های 1 و 2 کاملا بر پاسخ نهایی در این بخش منطبق اند و همگی منطبق با پاسخ محاسبه شده به صورت تئوری در بخش 1 هستند



شکل4 – محاسبه کانولوشن دو سیگنال داده شده در سوال 1 با استفاده تابع آماده conv در متلب

***

2.1 فیلترینگ و کرنل ها

1) در بحث پردازش تصویر، ما با تصاویر مانند ماتریس هایی برخورد می کنیم که هر درایه از ماتریس نمایندهٔ یک پیکسل از تصویر است. در برخی موارد بسته به هدف ما نیاز است فیلتر هایی روی عکس اعمال شوند. این فیلتر ها ممکن است تصویر را تار کنند (blurring) ، ممکن است آن را تشدید یا به اصطلاح تیز تر کنند (sharpening) ، ممکن است نیاز به تشخیص لبهٔ اشیاء درون تصویر باشد (edge detection) و یا کاربرد های مختلف دیگر. این عمل فیلتر کردن تصاویر با ماتریس های کوچک تری به نام kernel انجام می پذیرد. روش فیلتر کردن، یک نوع کانولوشن دو بعدی است؛ به این صورت که ماتریس کرنل روی تصویر اصلی شیفت داده می شود، درایه هایی که همپوشانی دارند، در هم ضرب می شوند سپس یک ماتریس جدید که ابعاد آن برابر با ابعاد ماتریس کرنل است، حاصل می شود؛ درایه های این ماتریس جمع زده می شوند و حاصل در درایهٔ مرکزی همپوشانی تصویر و کرنل قرار می گیرد. در نهایت یک ماتریس جدید حاصل می شود که فیلتر شدهٔ ماتریس اصلی است و به ما تصویر فیلتر شده را می دهد.

نمونه هایی از کرنل های معروف و کاربردی در زیر به همراه مثال آورده شده؛ در ابتدا یک تصویر اصلی داریم که آن را در ابتدای صفحهٔ بعد آورده ایم؛ سپس فیلتر های مختلف با استفاده از کرنل مخصوص آن ها روی تصویر اعمال شده و تصویر فیلتر شده را نمایش داده ایم.

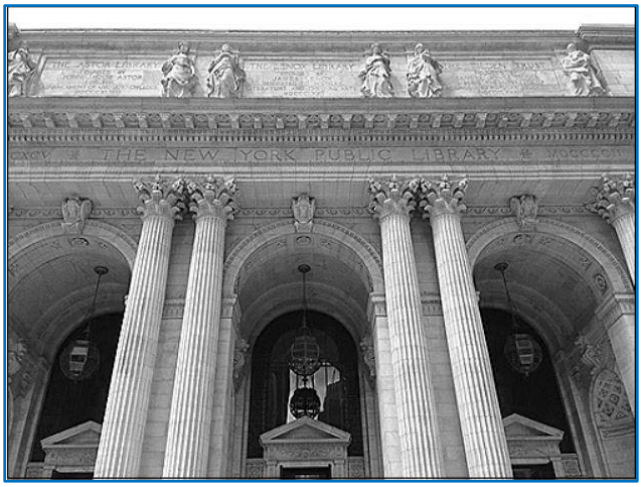
(منبع تصویر و کرنل ها : https://setosa.io/ev/image-kernels)

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

شکل5 – تصویر اصلی به همراه کرنل identity

کرنل identity ، کرنلی است که با اعمال شدن روی تصویر همان تصویر اصلی را به ما می دهد.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



شکل6 – تصویر فیلتر شده با استفاده کرنل sharpening

یک نوع خاص از فیلتر کردن تصاویر، تشدید یا به اصطلاح تیز کردن آن هاست؛ یک نوع کرنل مورد استفاده برای این کار در بالا آورده شده و تاثیر آن روی تصویر اصلی را می بینیم؛ همانطور که دیده می شود، اشیاء درون تصویر با وضوح بیشتری دیده می شوند؛ برای مثال نوشتهٔ "THE NEW YORK PUBLIC LIBRARY" روی سردر ورودی ساختمان قابل مشاهده است، در حالی که در تصویر اصلی به راحتی دیده نمی شد.

در صفحهٔ بعد یک نوع کرنل برای تار کردن تصویر (blurring) آورده شده؛ همانطور که دیده می شود، با اعمال این فیلتر با استفاده از کرنل نشان داده شده روی تصویر، خروجی تار و محو شده است.

$$\begin{bmatrix} 0.0625 & 0.125 & 0.0625 \\ 0.125 & 0.25 & 0.125 \\ 00.0625 & 0.125 & 0.0625 \end{bmatrix}$$

شکل7 – تصویر فیلتر شده با استفاده کرنل blurring

$$\begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

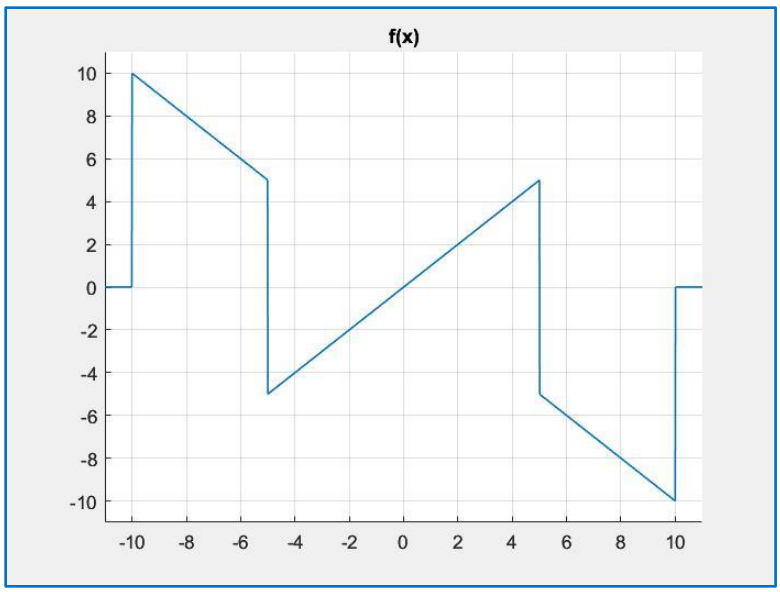
شکل8 – تصویر فیلتر شده با استفاده کرنل embossing

یک نوع دیگر از فیلتر کردن وجود دارد که به عنوان embossing معروف است و همان طور که از اسمش بر می آید، تصویر را برجسته می کند و توهم عمق داشتن به ما می دهد. یک نوع کرنل برای این کار در بالا آورده شده و تاثیر آن پس از اعمال فیلتر مربوطه روی تصویر نمایش داده شده است.

2) برای edge detection از کرنل های مختلفی استفاده می شود؛ بسته به این که بخواهیم لبه های عمودی یا افقی یا اریب اشیاء را تشخیص بدهیم کرنل مورد استفاده متفاوت خواهد بود؛ در زیر تعدادی کرنل مورد استفاده در edge detection را آورده‌ایم. کرنل سمت راست برای تشخیص لبه های عمودی و کرنل سمت چپ برای برای تشخیص لبه های افقی استفاده می شود.

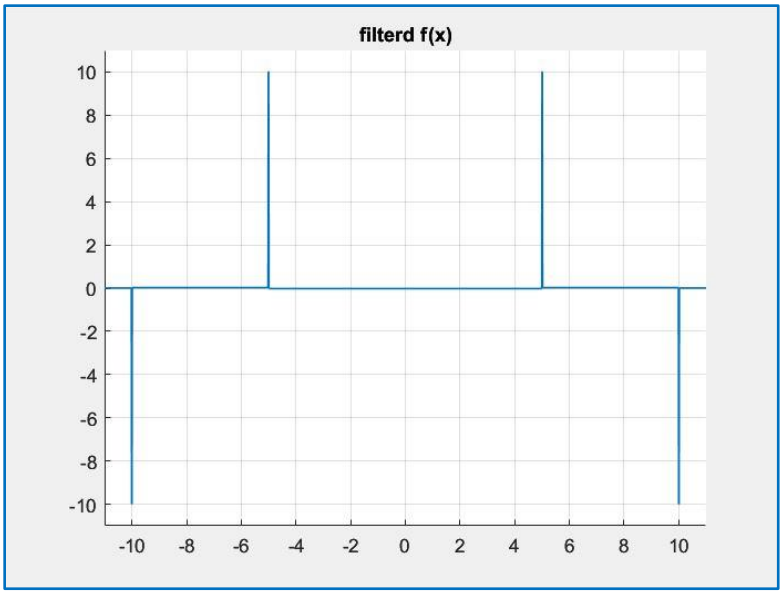
$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad ، \quad \begin{bmatrix} -1 & 0 & -1 \\ -2 & 0 & -2 \\ -1 & 0 & -1 \end{bmatrix}$$

3) ابتدا تابع f را رسم می کنیم؛ تصویر تابع در صفحهٔ بعد آورده شده است.



شکل9 – تابع f چند ضابطه ای داده شده در سوال

حال با استفاده از یک حلقه for روشی که در بخش یک برای اعمال فیلتر گفته شد را روی تابع اعمال می کنیم و تابع حاصل را در یک آرایه به نام filtered_f ذخیره و سپس آن را رسم می کنیم.

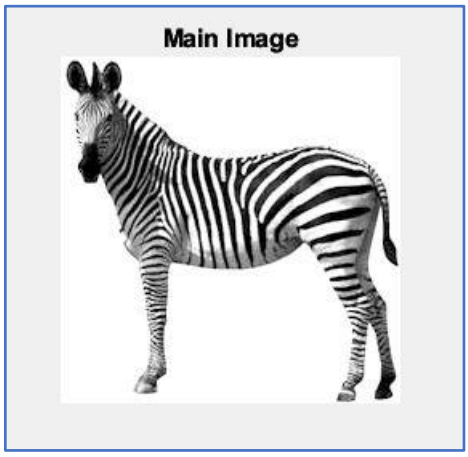


شکل10 – خروجی، پس اعمال فیلتر مربوطه روی تابع f

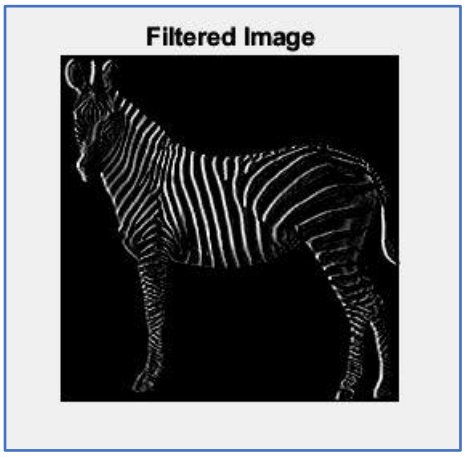
همانطور که دیده می شود، با اعمال فیلتر با استفاده از کرنل داده شده، می توان خطوط قائم در تابع اولیه را پیدا کرد.

4) حال باید کرنل داده شده را روی یک تصویر اعمال کنیم. ابتدا تصویر را در متلب لود می کنیم و سپس با استفاده از دستور imfliter ، آن را فیلتر می کنیم؛ این دستور یک ماتریس به عنوان ورودی اصلی و یک ماتریس به عنوان کرنل می گیرد و ماتریس اصلی را با استفاده از کرنل داده شده فیلتر می کند.

در زیر تصویر اصلی به همراه تصویر فیلتر شده، نمایش داده شده و تفاوت آن ها قابل مشاهده است.



شکل11 – تصویر اصلی بدون فیلتر



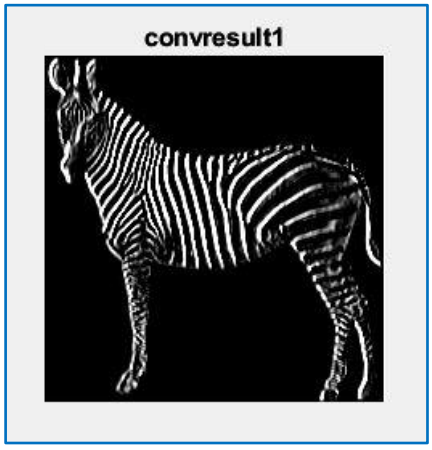
شکل12 – تصویر فیلتر شده با استفاده از کرنل یک بعدی

همانطور که دیده می شود در تصویر فیلتر شده لبه های عمودی (که همان خط های عمودی روی بدن گورخر هستند)، با وضوح بهتری نسبت به بقیه خط ها تشخیص داده شده اند.

5) از ظاهر کرنل های داده شده و همچنین کرنل های ارائه شده در بخش 1 و 2 می توان حدس زد که:

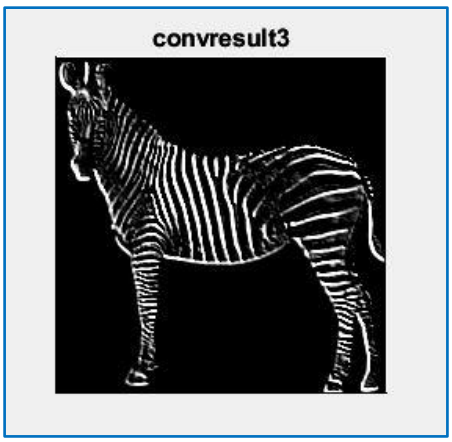
- A_1 : برای تشخیص لبه های عمودی به کار می رود.
- A_2 : برای تشخیص لبه های اریبی که در راستای قطر اصلی هستند به کار می رود.
- A_3 : برای تشخیص لبه های اریبی که در راستای قطر فرعی هستند به کار می رود.
- A_4 : برای تشخیص لبه های افقی به کار می رود.

6) حال در این بخش برای بررسی صحت حدس هایمان، با استفاده از دستور imfilter کرنل ها را روی تصویر اعمال می کنیم:

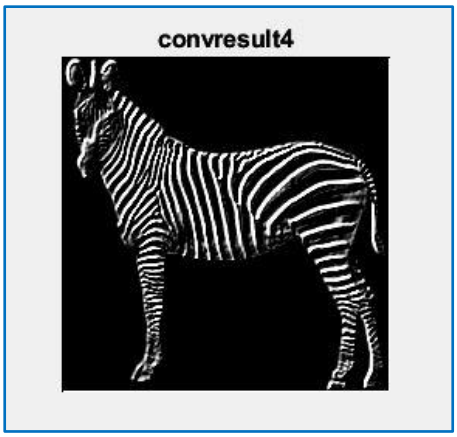


شکل13 – تصویر فیلتر شده با استفاده از کرنل خطوط عمودی

شکل14 – تصویر فیلتر شده با استفاده از کرنل خطوط افقی



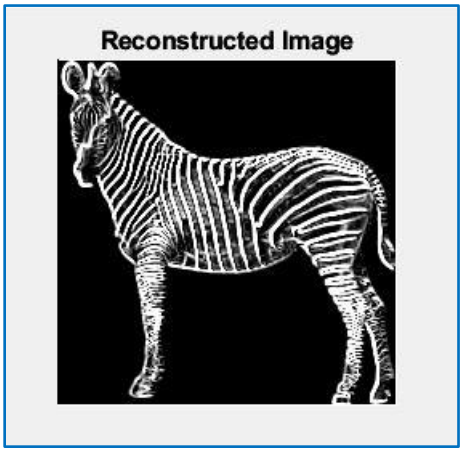
شکل15 – تصویر فیلتر شده با استفاده از کرنل خطوط قطری



شکل16 – تصویر فیلتر شده با استفاده از کرنل خطوط قطر فرعی

همانطور که دیده می شود، در هر شکل، قسمتی از خطوط روی بدن گورخر با وضوح بیشتری دیده می شود که در راستای کرنل اعمال شده به تصویر بوده است.

حال می خواهیم بین تصاویر بدست آمده یک ماکسیمم گیری انجام دهیم تا همه خطوط تشخیص داده شده، در تصویر جدید قابل نمایش باشند؛ برای انجام این کار دستور max را روی convresult1 و convresult2 اعمال می کنیم و در یک آرایهٔ جدید به نام semi_reconstructed_image1 ذخیره می کنیم؛ این دستور دو آرایه را دریافت می کند و با بررسی درایه به درایهٔ آن ها، مقدار ماکسیمم بین دو درایه را در یک درایهٔ متناظر در یک آرایه جدید قرار می دهد. مشابها روی convresult3 و convresult4 نیز یک بار این دستور را اجرا می کنیم و حاصل را در آرایهٔ جدیدی به نام semi_reconstructed_image2 ذخیره می کنیم. اکنون دوباره دستور max را روی دو آرایهٔ جدید بدست آمده اجرا می کنیم و در یک آرایهٔ نهایی به نام reconstructed_image ذخیره می کنیم و به صورت یک تصویر نمایش می دهیم؛ تصویر بدست آمده به صورت زیر است:



شکل17 – تصویر بدست آمده پس از بازسازی 4 تصویر اولیه

---

2 . تبدیل لاپلاس

1.2. آشنایی با تبدیل لاپلاس

1) در این بخش 4 سیگنال پیوسته در زمان به ما داده شده و از ما خواسته شده که تبدیل لاپلاس آن ها را حساب کنیم.

$$\boldsymbol{x_1(t) = (t\sin(t) + e^{-2t})u(t)}$$

با مراجعه به جدول تبدیل لاپلاس اطلاعات زیر را داریم:

$$\mathcal{L}\{sin(\alpha t)\} = \frac{\alpha}{s^2 + \alpha^2}$$

$$\mathcal{L}\{e^{\alpha t}\} = \frac{1}{s - \alpha}$$

همچنین از خواص تبدیل لاپلاس می دانیم:

$$\mathcal{L}\{tf(t)\} = -F'(s)$$

و به طور کلی تر:

$$\mathcal{L}\{t^n f(t)\} = (-1)^n F^{(n)}(s)$$

بنابراین تبدیل لاپلاس سیگنال $x_1(t)$ به صورت زیر خواهد بود:

$$X_1(s) = \frac{2s}{(s^2 + 1)^2} + \frac{1}{s + 2}$$

حال برای تایید صحت جوابمان، سیگنال داده شده را در متلب با استفاده از یک متغیر سمبولیک تعریف می کنیم و با استفاده از دستور laplace از آن تبدیل لاپلاس می گیریم و در command window آن را دریافت می کنیم:

```
1/(s + 2) + (2*s)/(s^2 + 1)^2
```

شکل18 – تبدیل لاپلاس سیگنال $x_1(t)$ در متلب

$$\boldsymbol{x_2(t) = (t^2 e^{-5t})u(t)}$$

با توجه به اطلاعاتمان از جدول تبدیل لاپلاس و خواص تبدیل لاپلاس که در بالا آورده شد، داریم:

$$X_2(s) = \frac{2}{(s + 5)^3}$$

در حقیقت دوبار از تبدیل لاپلاس e^{-5t} مشتق گرفتیم.

حال سیگنال را در متلب تعریف کرده و از آن تبدیل لاپلاس می گیریم:

```
2/(s + 5)^3
```

شکل19 – تبدیل لاپلاس سیگنال $x_2(t)$ در متلب

سیگنال سومی که به ما داده شده به صورت زیر است:

$$\boldsymbol{x_3(t) = \cos(2t)\, u(t)}$$

با مراجعه به جدول تبدیل لاپلاس داریم:

$$\mathcal{L}\{\cos(\alpha t)\} = \frac{s}{s^2 + \alpha^2}$$

بنابراین تبدیل لاپلاس سیگنال $x_3(t)$ به صورت زیر خواهد بود:

$$X_3(s) = \frac{s}{s^2 + 4}$$

پاسخ بدست آمده در متلب نیز محاسبات ما را تایید می کند:

```
s/(s^2 + 4)
```

شکل20 – تبدیل لاپلاس سیگنال $x_3(t)$ در متلب

$$\boldsymbol{x_4(t) = t^2 \sinh(2t)\, u(t)}$$

با مراجعه به جدول تبدیل لاپلاس داریم:

$$\mathcal{L}\{\sinh(\alpha t)\} = \frac{\alpha}{s^2 - \alpha^2}$$

بنابراین تبدیل لاپلاس سیگنال $x_4(t)$ برابر با مشتق دوم تبدیل لاپلاس تابع $\sinh(2t)$ خواهد بود؛ پس خواهیم داشت:

$$X_4(s) = \frac{-4}{(s^2 - 4)^2} + \frac{16s^2}{(s^2 - 4)^3}$$

محاسبات متلب نیز نشان می دهد که جواب ما درست است:

```
(16*s^2)/(s^2 - 4)^3 - 4/(s^2 - 4)^2
```

شکل21 – تبدیل لاپلاس سیگنال $x_4(t)$ در متلب

2) در این بخش، تبدیل لاپلاس تعدادی سیگنال به ما داده شده است و ما باید با محاسبه وارون تبدیل لاپلاس، خود سیگنال ها را بدست آوریم.

$$\boldsymbol{F_1(s) = \frac{1}{s(s+2)(s+3)}}$$

با تجزیه تابع داده شده به کسر های جزئی داریم:

$$F_1(s) = \frac{A}{s} + \frac{B}{s+2} + \frac{C}{s+3}$$

این معادله منتهی به یک دستگاه سه معادله سه مجهول خواهد شد که با حل آن داریم:

$$A = \frac{1}{6}\,, B = \frac{-1}{2}\,, C = \frac{1}{3}$$

می دانیم:

$$\mathcal{L}^{-1}\left\{\frac{1}{s+\alpha}\right\} = e^{-\alpha t}u(t)$$

بنابراین وارون تبدیل لاپلاس تابع داده شده به صورت زیر خواهد بود:

$$f_1(t) = \left(\frac{1}{6} - \frac{1}{2}e^{-2t} + \frac{1}{3}e^{-3t}\right)u(t)$$

اکنون تابع داده شده را در متلب با استفاده از یک متغیر سمبولیک تعریف می کنیم و با استفاده از دستور ilaplace وارون تبدیل لاپلاس آن را محاسبه و در command window آن را دریافت می کنیم:

```
exp(-3*t)/3 - exp(-2*t)/2 + 1/6
```

شکل22 – وارون تبدیل لاپلاس تابع $F_1(s)$ در متلب

$$\boldsymbol{F_2(s) = \frac{10}{(s+1)^2(s+3)}}$$

با تجزیهٔ تابع به کسر های جزئی داریم:

$$F_2(s) = \frac{A}{s+1} + \frac{B}{(s+1)^2} + \frac{C}{s+3}$$

این معادله منتهی به یک دستگاه سه معادله سه مجهول خواهد شد که با حل آن داریم:

$$A = -2.5\,, B = 5\,, C = 2.5$$

می دانیم:

$$\mathcal{L}^{-1}\left\{\frac{n!}{(s+\alpha)^{n+1}}\right\} = t^n e^{-\alpha t}u(t)$$

بنابراین وارون تبدیل لاپلاس تابع داده شده برابر خواهد شد با:

$$f_2(t) = (-2.5e^{-t} + 5te^{-t} + 2.5e^{-3t})u(t)$$

محاسبات متلب نیز صحت جواب ما را تایید می کند:

```
(5*exp(-3*t))/2 - (5*exp(-t))/2 + 5*t*exp(-t)
```

شکل23 – وارون تبدیل لاپلاس تابع $F_2(s)$ در متلب

$$F_3(s) = \frac{2s^2}{(s^2+1)(s-1)^2}$$

با تجزیه تابع به کسر های جزئی داریم:

$$F_3(s) = \frac{As+B}{s^2+1} + \frac{C}{s-1} + \frac{D}{(s-1)^2}$$

این معادله به یک دستگاه 4 معادله 4 مجهول منتهی خواهد شد که با حل آن داریم:

$$A = -1\,, B = 0\,, C = 1\,, D = 1$$

یعنی :

$$F_3(s) = \frac{-s}{s^2+1} + \frac{1}{s-1} + \frac{1}{(s-1)^2}$$

از آن جایی که اولین جمله در سمت چپ معادله تبدیل لاپلاس $-\cos t$ است، داریم:

$$f_3(t) = (-\cos t + e^{-t} + te^{-t})u(t)$$

در متلب نیز جواب مشابه را بدست می آوریم:

```
exp(t) - cos(t) + t*exp(t)
```

شکل24 – وارون تبدیل لاپلاس تابع $F_3(s)$ در متلب

***

2.2. تحلیل سیستم با استفاده از تبدیل لاپلاس

1) در این بخش تعدادی تابع سیستم داده شده و باید پایداری آن ها را بررسی کنیم.

شیوه کار ما به این صورت است که با استفاده از دستور tf تابع تبدیل سیستم را در متلب توصیف می کنیم، سپس با استفاده از دستور pzplot نمودار صفر و قطب سیستم را رسم کرده و در مورد پایداری آن سیستم نظر می دهیم؛ می دانیم برای این که یک سیستم علی، پایدار باشد، تمام قطب های آن باید در سمت چپ صفحه مختلط قرار بگیرند. (دستور tf، ضرایب چند جمله ای در صورت و مخرج در تابع سیستم را می گیرد و ثبت می کند.)

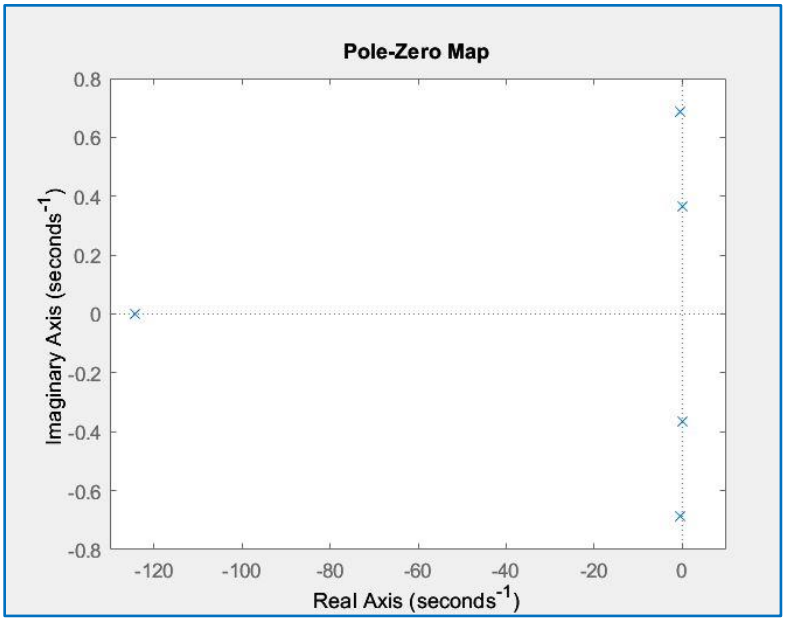
همچنین یک روش دوم نیز در این بخش در پیش گرفتیم؛ برای هر سیستم یک متغیر Boolean تعریف کرده ایم و با استفاده از دستور isstable پایداری آن را بررسی کرده ایم؛ این دستور تابع تبدیل توصیف شده به کمک دستور tf را به عنوان ورودی می گیرد و اگر سیستم پایدار بود، 1 و در غیر این صورت 0 بر می گرداند.

اولین سیستمی که آن را بررسی می کنیم به صورت زیر است:

```
                                1
-----------------------------------------------------
s^5 + 125 s^4 + 100 s^3 + 100 s^2 + 20 s + 10
```

شکل25- تابع سیستم $H_1(s)$ توصیف شده با استفاده از دستور tf

نمودار صفر و قطب این سیستم به شکل زیر است:



شکل26- نمودار صفر و قطب سیستم $H_1(s)$

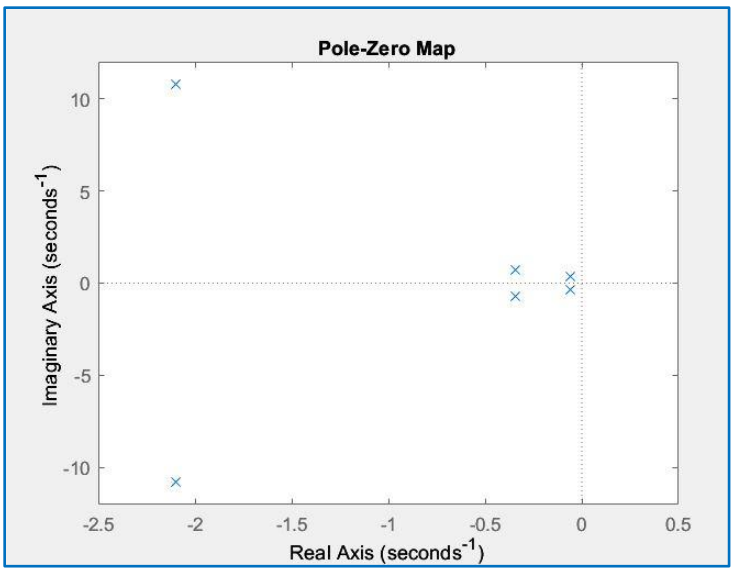
همانطور که دیده می شود، قطب ها در سمت چپ صفحهٔ مختلط هستند. بنابراین می توان گفت این سیستم پایدار است.

دومین سیستمی که داده شده، به صورت زیر توصیف می شود:

```
                                1
-----------------------------------------------------------
s^6 + 5 s^5 + 125 s^4 + 100 s^3 + 100 s^2 + 20 s + 10
```

شکل27- تابع سیستم $H_2(s)$ توصیف شده با استفاده از دستور tf

نمودار صفر و قطب این سیستم در صفحهٔ بعد آورده شده است:



شکل28- نمودار صفر و قطب سیستم $H_2(s)$

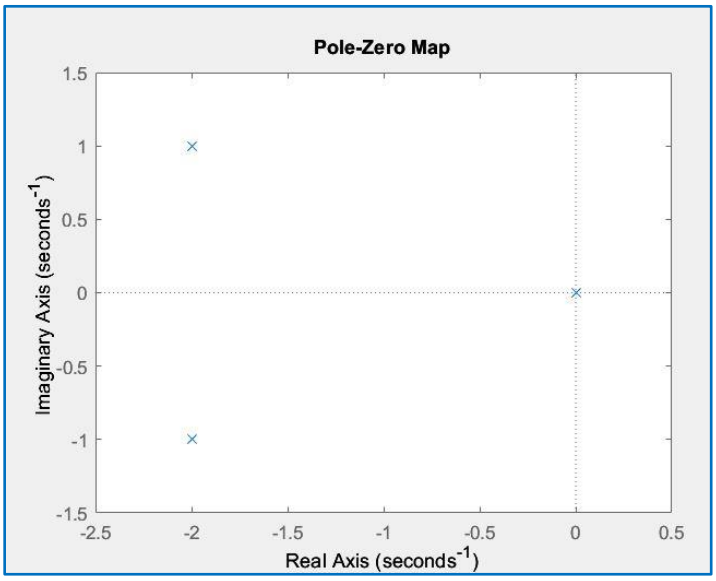
قطب های این سیستم نیز همگی در سمت چپ صفحهٔ مختلط هستند؛ بنابراین این سیتم نیز پایدار است.

سیستم آخری که به ما داده شده به صورت زیر توصیف می شود:

```
          5
-----------------
s^3 + 4 s^2 + 5 s
```

شکل29- تابع سیتسم $H_3(s)$ توصیف شده با استفاده از دستور tf

نمودار صفر و قطب این سیتسم به شکل زیر است:



شکل30- نمودار صفر و قطب سیستم $H_3(s)$

با وجود این که قطب ها در سمت چپ صفحهٔ مختلط هستند، اما از آنجایی که یک قطب در صفر داریم، این سیستم پایدار نمی باشد.

```
H1 is stable
H2 is stable
H3 is unstable
```

شکل31 – بررسی پایداری سیستم های داده شده با استفاده از دستور isstable

2) می دانیم اگر تابع یک سیستم به صورت زیر داده شده باشد:

$$H(s) = \frac{Y(s)}{X(s)}$$

که در آن X تبدیل لاپلاس ورودی و Y تبدیل لاپلاس خروجی است، آنگاه:

$$Y(s) = H(s)X(s)$$

در نتیجه:

$$y(t) = \mathcal{L}^{-1}\{H(s)X(s)\}$$

با در پیش گرفتن این روش و استفاده از دستور های laplace و ilaplace در متلب، پاسخ این سیستم به هر یک از ورودی های داده شده را بدست می آوریم.

```
2*exp(-t) - 3*exp(-t/2) + 1
```

شکل32- پاسخ سیستم به ورودی $x_1(t)$

```
(21*cos(3*t))/185 - (3*exp(-t))/5 + (18*exp(-t/2))/37 - (22*sin(3*t))/185
```

شکل33- پاسخ سیستم به ورودی $x_2(t)$

```
4*exp(-t) - 4*exp(-t/2) + (3*t*exp(-t/2))/2
```

شکل34- پاسخ سیستم به ورودی $x_3(t)$

دقت شود که در همهٔ عبارات بالا باید یک u(t) ضرب شود. (دستورهای laplace و ilaplace تابع پله را به طور پیشفرض در نظر می گیرند.)

3.2. حل معادله با استفاده از تبدیل لاپلاس

معادله دیفرانسیل داده شده به صورت زیر است:

$$\frac{dy}{dt} + 2y = 12e^{3t}\,, y(0) = 3$$

می دانیم:

$$\mathcal{L}\left\{\frac{df}{dt}\right\} = sF(s) - f(0)$$

با توجه به این مطلب، از دو طرف معادلهٔ داده شده، تبدیل لاپلاس می گیریم:

$$sY(s) - 3 + 2Y(s) = \frac{12}{s-3}$$

حال این معادله را به متلب می دهیم تا Y(s) را برحسب s بدست بیاورد؛ سپس با استفاده از دستور ilaplace وارون تبدیل لاپلاس آن را حساب می کنیم تا پاسخ معادله بدست آید:

```
(3*exp(-2*t))/5 + (12*exp(3*t))/5
```

شکل35 – پاسخ معادله دیفرانسیل داده شده

مجددا دقت شود که در عبارت بالا باید یک u(t) ضرب شود. (دستورهای laplace و ilaplace تابع پله را به طور پیشفرض در نظر می گیرند.)

---

3 . تولید نت های موسیقی با استفاده از یک سیستم گسسته

1.3. الگوریتم Karplus-Strong

در ابتدا باید خاطر نشان کرد که صوت، یک پدیده پیوسته در زمان است، اما در کامپیوتر ما آن را به صورت گسسته شبیه سازی می کنیم.

نحوه شبیه سازی به این صورت است که یک نمونه ورودی به طول N داریم (که N همان دوره تناوب سیگنال صوتی ماست) که این نمونه می تواند به فرم سینوسی، مثلثی، پله ای، و هر فرم دلخواه دیگری باشد؛ سپس این نمونه را به عنوان ورودی به سیستمی می دهیم که بلوک دیاگرام آن در فایل تمرین آمده است و در خروجی، سیگنال صوتی ما آماده است.

یکی از الگوریتم های شبیه سازی صوت، الگوریتم Karplus-Strong است که برای شبیه سازی صدای سیم های گیتار به کار می‌رود. در این الگوریتم، ورودی ما یک نمونه از اعداد تصادفی بین 1- و 1 است که با توزیع یکنواخت تولید شده اند.

2.3. تولید یک قطعه موسیقی

با توجه به مراحل گفته شده در سوال توابع generateNote و noteFreq را می نویسیم؛ در تابع generateNote سیگنال y را به طور دلخواه 50 برابر duration هر نوت در نظر می گیریم و ضابطهٔ مربوط به دیاگرام بلوکی داده شده را پیاده سازی می کنیم. در نهایت با نوشتن تابع getSong آرایهٔ نوت ها به همراه مدت زمان آن ها را دریافت کرده و تمام نوت ها را در یک آرایه جمع آوری می کنیم و در آخر با استفاده از دستورات audioplayer و play آن را پخش می کنیم.