بخش ۱: آشنایی با متلب

توضیح مثال اول:

این قطعه کد، یک سیگنال تولید می‌کند و آن را رسم می‌کند. برای بررسی دقیق تر به توضیح هر خط کد می‌پردازیم:

Fs=100 نشان دهنده فرکانس نمونه برداری است. نشان می‌دهد در هر ثانیه چند نمونه گرفته می‌شود.

t=-30:1/fs:30 با این خط کد یک بردار زمان می‌سازیم که از ‎-۳۰‎ تا ۳۰ را شامل می‌شود. گام اضافه شدن آن برابر 1/fs می‌باشد و از آنجایی که فرکانس نمونه برداری ما ۱۰۰ است بنابراین در این بردار زمان ۰.۰۱ ثانیه زیاد می‌شود.

x = exp(-20.*abs(t)) در اینجا سیگنال مورد نظر را می‌سازیم. در واقع x سیگنال تولیدی است. این سیگنال در واقع exponential قدر مطلق بردار زمان ضربدر ‎-۲۰‎ می‌باشد (همان طور که در صورت مسئله هم اشاره شده، قصد رسم این تابع را داریم). نکته قابل توجه در این خط علامت .* است که نشان دهنده ضرب عنصر به عصنر (element-wise multiplication) است.

plot(t,x) این خط نمودار سیگنال x را برحسب بردار زمان رسم می‌کند.

سایر خطوط ویژگی های نمودار رسم شده، از جمله حد محور x و y، نام محور ها، نام نمودار و... را تعیین می‌کنند.

توضیح مثال دوم:

این قطعه کد همزمان چند سیگنال را در یک نمودار رسم می‌کند. برای بررسی دقیق تر به توضیح هر خط کد می‌پردازیم:

plot(t3,P_r_Beta_0) سیگنال P_r_Beta_0 را نسبت به t3 رسم می‌کند.

hold on در واقع به متلب اعلام می‌کنیم که نمودار فعلی را با تمام ویژگی هایش نگه دارد تا بتوانیم سیگنال جدید را روی آن رسم کنیم. در صورتی که از این خط استفاده نکنیم، با اضافه شدن یک سیگنال جدید، سیگنال قبلی از روی نمودار پاک می‌شود و سیگنال جدید جایگزین آن می‌شود.

plot(t3,P_r2_Beta_0) سیگنال دوم را رسم می‌کند.

plot(t3,P_r3_Beta_0) سیگنال سوم را رسم می‌کند.

legend("Ideal Sampling ', 'Sampling Error = 0.1T', 'Sampling Error = 0.2T') برای هر سیگنال اسم می‌گذارد و آن را در نمودار کشیده شده نشان می‌دهد. اسم ها به ترتیبی که در خط زده‌ایم به سیگنال ها از ابتدا تا انتها اختصاص داده می‌شوند.

سایر خطوط ویژگی های نمودار را تعیین می‌کنند.

hold off این خط به متلب می‌فهماند که در صورتی که قصد کشیدن سایر سیگنال ها را داریم، نمودار فعلی را پاک کند و سیگنال‌های جدید را جایگزین آن کند.

بخش ۱.۱ رسم نمودار:

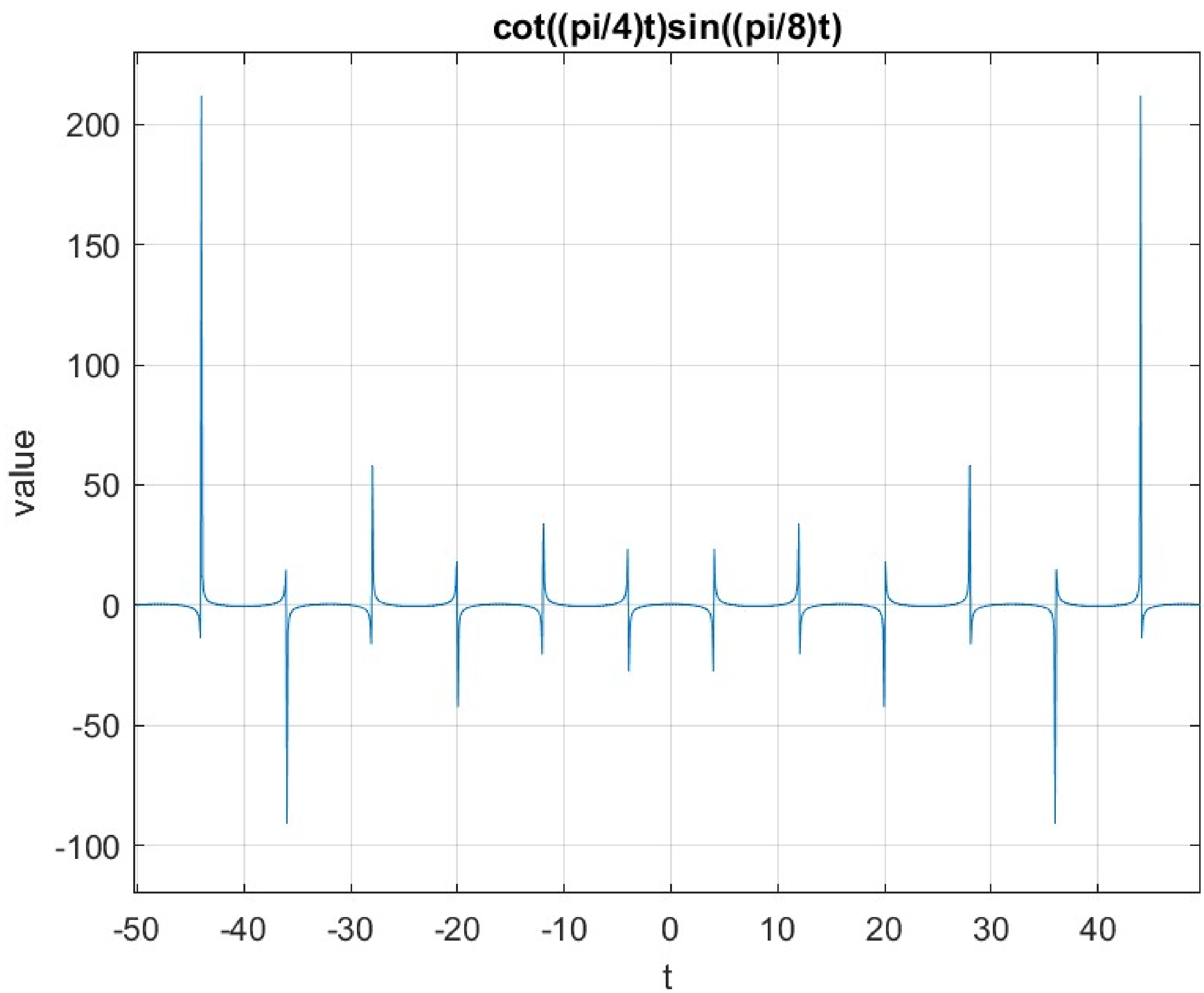
برای رسم توابع داده شده مانند مثال‌های قبلی عمل می‌کنیم. ابتدا بازه محور x را تعیین کرده و سپس با توجه به نوع نمودار، به ازای هر کدام از مقادیر مجاز مقدار خروجی تابع را محاسبه کرده و رسم می‌کنیم.

<u>کد مربوط به تابع اول: part_1_1_1</u>

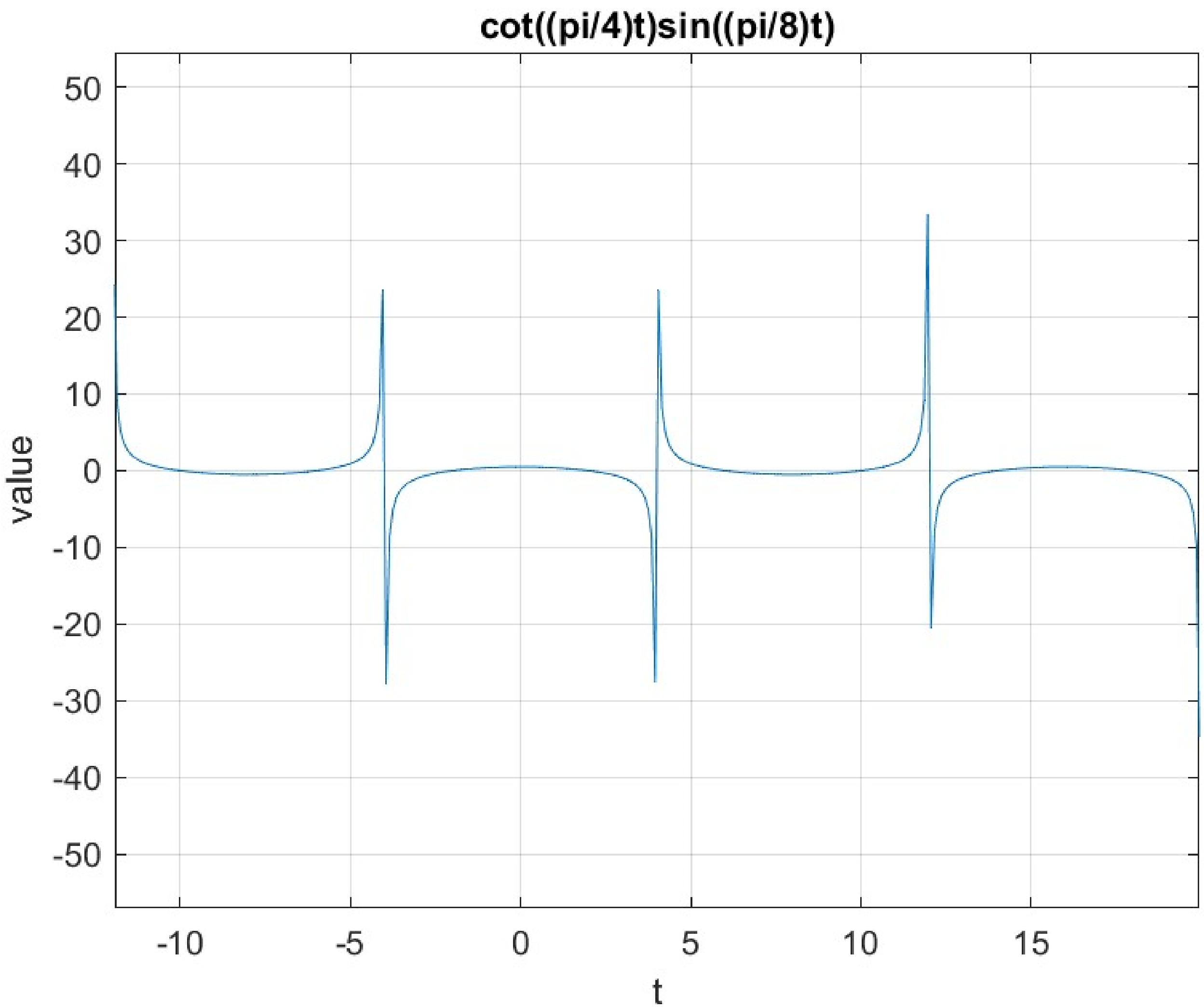
در این قطعه کد سعی داریم تابع داده شده را در بازه [50,50-] رسم کنیم. بنابراین ابتدا بازه محور x را مشخص می‌کنیم. سپس توابع کتانژانت و سینوس را جداگانه تعریف کرده در درنهایت در هم ضرب می‌کنیم. چون متلب با ماتریس‌ها کار می‌کند باید از *. استفاده کنیم که بتواند بُعد ها را تطبیق دهد. تصویر زیر قطعه کد این بخش را نشان می‌دهد.

```
t = linspace(-50, 50, 1000);
sin_value = sin((pi/8)*t);
cot_value = cot((pi/4)*t);
y = cot_value.*sin_value;
plot(t, y);
xlabel('t');
ylabel('value');
title('cot((pi/4)t)sin((pi/8)t)');
grid on;
```

با اجرای این کد خروجی نمودار به شکل زیر خواهد شد:



برای بررسی دقیق‌تر قسمتی از بازه رسم شده را بزرگ‌نمایی می‌کنیم:

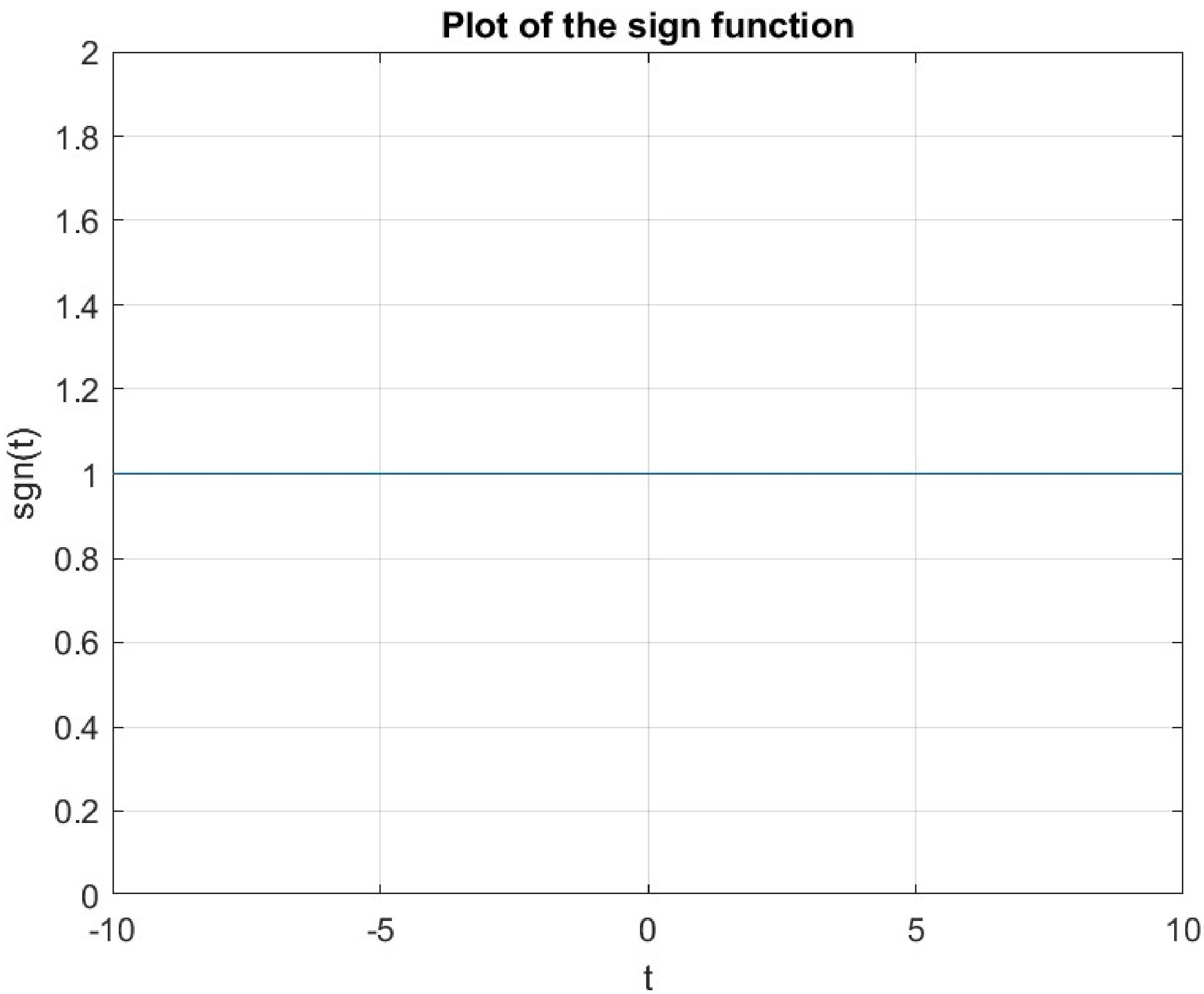


کد مربوط به تابع دوم: part_1_1_2

در این قطعه کد می‌خواهیم تابع علامت را برای بازه [-10,10] رسم کنیم. ابتدا این بازه را تعریف می‌کنیم. و سپس به کمک تابع sign خروجی تابع را محاسبه می‌کنیم. توجه شود چون متلب با ماتریس کار می‌کند، برای تطبیق بُعد ها باید قبل از علائم محاسبه‌گر مثل تقسیم (/) یا توان (^) نقطه بگذاریم. این قطعه کد به شکل زیر خواهد بود:

```
t = -10:1:10;
y = sign(1./(t.^2));
plot(t, y);
xlabel('t');
ylabel('sgn(t)');
title('Plot of the sign function');
grid on;
```

چون این تابع، تابع علامت است خروجی آن دو حالت دارد: -۱ و ۱. برای اعداد مثبت خروجی آن ۱ و برای اعداد منفی خروجی آن -۱ خواهد بود. با اجرای این قطعه کد خروجی نمودار به شکل زیر خواهد بود:



با توجه به اینکه در ورودی تابع sgn مقدار t را به توان دو رسانده‌ایم، بنابراین هیچ وقت به آن عدد منفی نداده‌ایم که خروجی آن ۱- شود. طبق نمودار هم خروجی این تابع به ازای تمام مقادیر ۱ خواهد بود.

<u>کد مربوط به تابع سوم: part_1_1_3</u>

در این قطعه مد سعی داریم تا یک تابع چند ضابطه‌ای رسم کنیم. ضابطه اول یک تابع ثابت است. برای رسم تابع ثابت در متلب از ones(size(t)) استفاده می‌کنیم. برای سادگی در نمایش، بازه‌ای که این تابع را برای آن بررسی می‌کنیم، بازه [-10,10] می‌باشد. برای مقادیر کوچکتر از ۳- از ضابطه اول استفاده می‌کنیم.

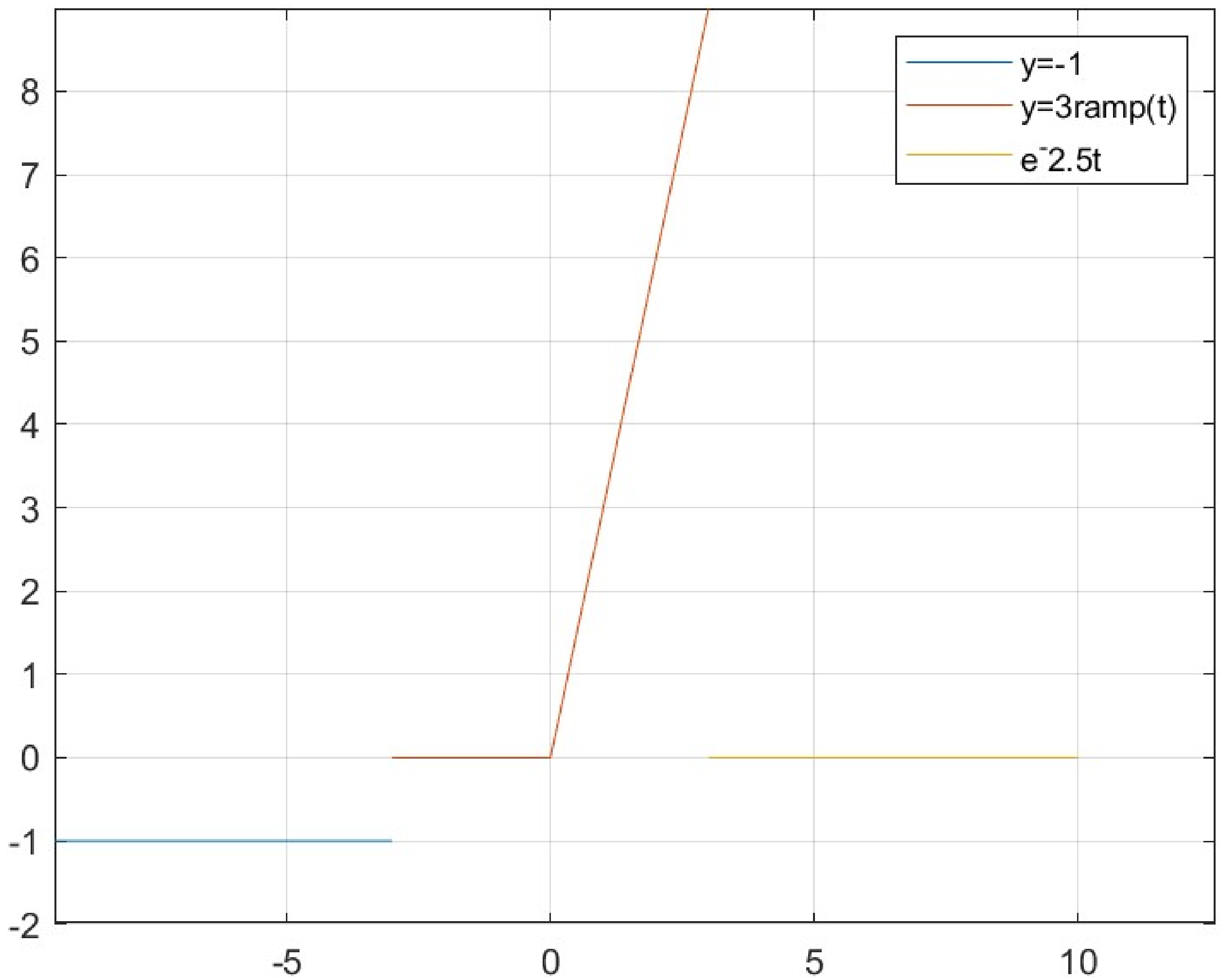
ضابطه دوم تابع ramp یا شیب است. این تابع به طور تعریف شده در متلب وجود ندارد. بنابراین برای رسم آن باید بر اساس اطلاعی که از آن داریم آن را رسم کنیم. می‌دانیم تابع شیب به شکل زیر تعریف می‌شود:

$$r(t) = \begin{cases} 0, & t \leq 0 \\ t, & t \geq 0 \end{cases}$$

بنابراین برای مقادیر کوچکتر از صفر، خروجی آن صفر و برای مقادیر بزرگتر از صفر همان مقدار ورودی را خروجی می‌دهیم. برای سادگی و جلوگیری از نوشتن توابع به صورت شرطی، خروجی تابع شیب را به شکل max(0.t) تعریف می‌کنیم که برای مقادیر منفی صفر و برای مقادیر مثبت خود آنها را خروجی می دهد. سپس آن را در ۳ ضرب می‌کنیم.

ضابطه سوم یک تابع exponential است. این تابع به صورت آماده و با نام exp در متلب تعریف شده است که از آن برای محاسبه استفاده می‌کنیم.

با توجه به اینکه قصد داریم تمامی این توابع در یک نمودار رسم شوند، باید از hold on استفاده کنیم. همچنین با استفاده از legend برای هر قسمت از این تابع چند ضابطه ای، اسم تعیین می‌کنیم. پس از اجرای کد، خروجی به شکل زیر خواهد بود:



توجه شود برای ضابطه سوم یعنی تابع exponential، مقادیر تقریبا به صفر میل می‌کنند.

بخش ۲: سری فوریه

بخش ۲.۱: محاسبه سری فوریه:

کد مربوط به این قسمت: FourierSeriesCalculator1 و FourierSeriesCalculator1_run

در این قسمت قصد داریم تابعی بنویسیم که سری فوریه توابعی به فرم x^a را محاسبه کند. می‌دانیم برای محاسبه سری فوریه توابع، از فرمول‌های زیر باید استفاده کنیم:

$$a_0 = \frac{1}{2L}\int_{-L}^{L} f(x)dx$$

$$a_n = \frac{1}{L}\int_{-L}^{L} f(x)\cos(\frac{n\pi}{L}x)dx$$

$$b_n = \frac{1}{L}\int_{-L}^{L} f(x)\sin(\frac{n\pi}{L}x)dx$$

$$f(x) = a_0 + \sum_{n=1}^{\infty} a_n\cos(nx) + b_n\sin(nx)$$

حال همین فرمول‌ها را در تابعی در متلب پیاده‌سازی می‌کنیم. تابع به فرم زیر خواهد بود:

```
function [f, t] = FourierSeriesCalculator1(Num, P, a, Nshow)
t = linspace(-2*P, 2*P, 1000);
a0 = (1/(P)) * integral(@(x) x.^a, -P, P);
disp('a0')
disp(a0)
an = zeros(Num, 1);
bn = zeros(Num, 1);
for n = 1:Num
    an(n) = (1/P) * integral(@(x) x.^a .* cos((pi*n*x)/P), -P, P);
    bn(n) = (1/P) * integral(@(x) x.^a .* sin((pi*n*x)/P), -P, P);
end
disp('an')
disp(an)
disp('bn')
disp(bn)
f = (a0/2);
for n = 1:Nshow
    f = f + an(n)*cos(pi*n*t/P) + bn(n)*sin(pi*n*t/P);
end
end
```

در خط اول تعریف تابع مورد نظر را می‌نویسیم. این تعریف تعداد ورودی‌های تابع و خروجی‌های آن را مشخص می‌کند. ورودی‌های تابع مطابق دستور داده شده است. در خروجی تابع، f نمایش سری فوریه تابع و t همان محور x یا همان بردار زمانی است.

در ابتدا برای محاسبه سری فوریه باید تناوب را مشخص کنیم. با توجه به P بازه‌ای که باید محاسبه کنیم [-2p,2p] خواهد بود. در این بازه ۱۰۰۰ نمونه یا نقطه داریم. پس از آن به محاسبه ضرایب سری فوریه مطابق فرمول های بالا می‌پردازیم. ابتدا ضریب a_0 را محاسبه می‌کنیم که همان DC است. با توجه به فرمول کافی است حاصل انتگرال محاسبه شود. انتگرال و بازه آن را در تابع آماده متلب برای محاسبه انتگرال جای‌گذاری کرده و خروجی آن را در متغیری به اسم a_0 قرار می‌دهیم و به کمک تابع disp این مقدار محاسبه شده را چاپ می‌کنیم.

برای محاسبه ضرایب سینوسی و کسینوسی سری فوریه، ابتدا باید دو بردار را مقداردهی اولیه کنیم. بردار a_n برای ضرایب کسینوسی و بردار b_n برای ضرایب سینوسی. با توجه به اینکه تعداد جملات سری فوریه یکی از ورودی های تابع است، اندازه این دو بردار برابر با همان تعداد جملات یعنی Num خواهد بود.

پس از ساخت بردار ها، به کمک حلقه for هر کدام از ضرایب را محاسبه می‌کنیم. برای محاسبه ضرایب سینوسی و کسینوسی به کمک انتگرال به ازای هر مقدار n از فرمول های بالا استفاده می‌کنیم.

در نهایت پس از خارج شدن از حلقه، مقادیر محاسبه شده برای ضرایب سینوسی و کسینوسی را چاپ می‌کنیم.

پس محاسبه ضرایب به کمک حلقه دوم نمایش سری فوریه تابع را می‌سازیم به گونه‌ای که ابتدا f را با مقدار $\frac{a_0}{2}$ مقدار دهی اولیه می‌کنیم و سپس جملات را به آن اضافه می‌کنیم. چون می‌دانیم طبق فرمول های بالا یک سیگما وجود دارد. این سیگما را به کمک حلقه for پیاده‌سازی کردیم.

در کد بعدی به کمک همین تابعی که نوشتیم، سری فوریه تابع x^3 را محاسبه می‌کنیم و آن را رسم می‌کنیم.

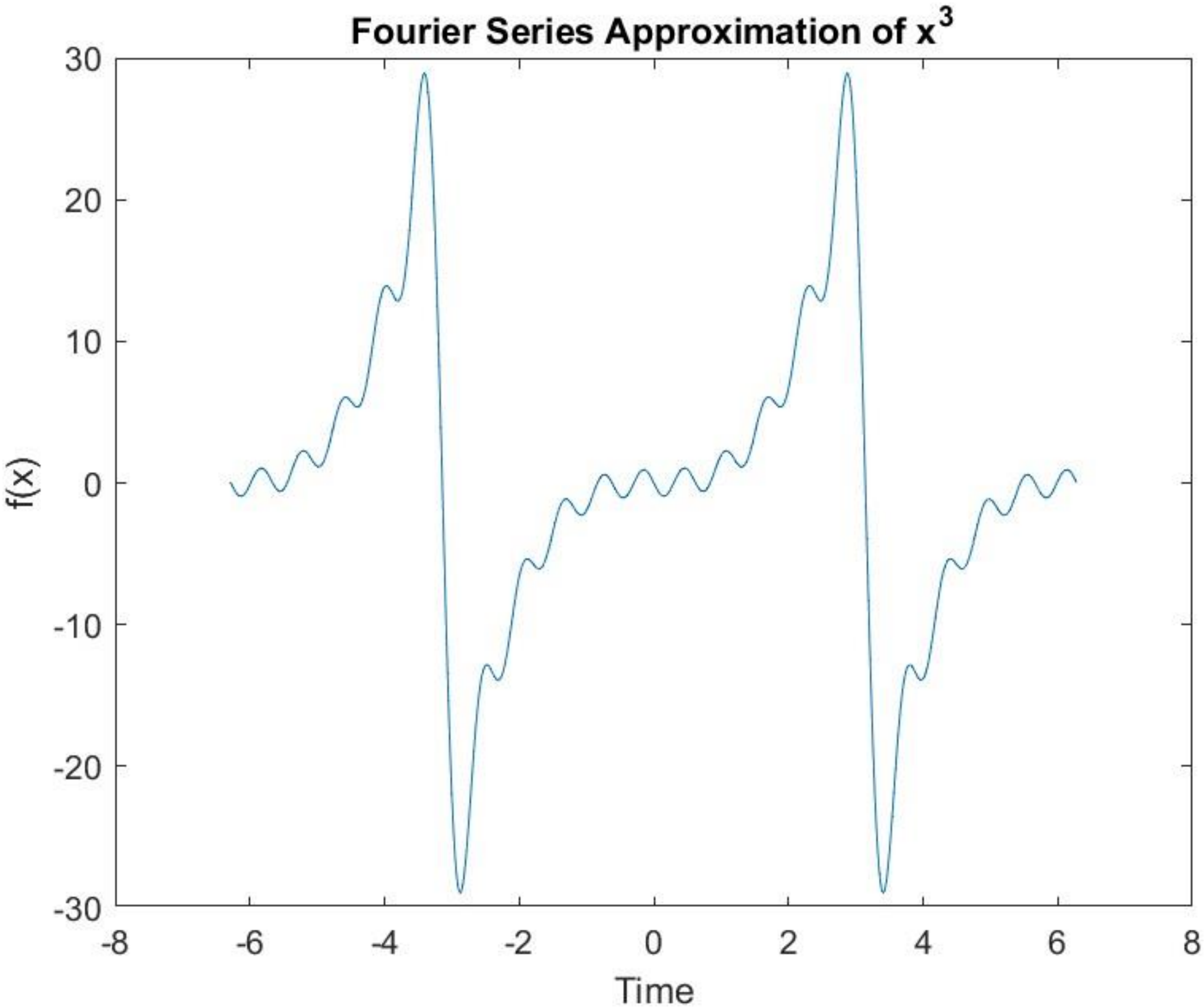
```
Num = 10;
P = pi;
a = 3;
Nshow = 10;
[f, t] = FourierSeriesCalculator1(Num, P, a, Nshow);
plot(t, f);
xlabel('Time');
ylabel('f(x)');
title('Fourier Series Approximation of x^3');
```

در این کد، سری فوریه تابع x^3 محاسبه می‌شود. در ترمینال ضرایب سری فوریه چاپ می‌شوند و در نهایت فرم سری فوریه تابع را رسم می‌کنیم. پس اجرای کد، خروجی‌ها به شکل زیر خواهند بود:

خروجی در ترمینال:

```
>> FourierSeriesCalculator1_run
a0
   5.6543e-16

an
   1.0e-14 *

   -0.1696
    0.1696
   -0.0751
    0.0848
   -0.0777
   -0.0212
    0.2898
   -0.5160
    0.2756
    0.0919

bn
    7.7392
   -8.3696
    6.1353
   -4.7473
    3.8518
   -3.2343
    2.7849
   -2.4440
    2.1768
   -1.9619
```

نمودار سری فوریه محاسبه شده:



بخش ۲.۲: محاسبه سری فوریه تابع خاص

<u>کد های این قسمت: FourierSeriesCalculator2 و FourierSeriesCalculator2_run</u>

برای محاسبه سری فوریه این تابع خاص لازم است تغییراتی در تابع قسمت قبلی اعمال کنیم چراکه تابع مورد نظر تغییر کرده و نمی‌توان با شکل قبلی آن را محاسبه کرد. تنها تغییری که لازم است اعمال کنیم، تغییر دادن تابع درون انتگرال است. همچنین یک پارامتر دیگر به عنوان ورودی تابع در نظر می‌گیریم که در واقع ضریب x داخل تابع ln می‌باشد. پس از اعمال تغییرات، تابع نهایی به فرم زیر خواهد بود:

```
function [f, t] = FourierSeriesCalculator2(Num, P, a, b , Nshow)
t = linspace(-2*P, 2*P, 1000);
a0 = (1/(P)) * integral(@(x) x.^b .* log(a.*x), -P, P);
disp('a0')
disp(a0)
an = zeros(Num, 1);
bn = zeros(Num, 1);
for n = 1:Num
    an(n) = (1/P) * integral(@(x) x.^b .* log(a.*x) .* cos((pi*n*x)/P), -P, P);
    bn(n) = (1/P) * integral(@(x) x.^b .* log(a.*x) .* sin((pi*n*x)/P), -P, P);
end
disp('an')
disp(an)
disp('bn')
disp(bn)
f = (a0/2);
for n = 1:Nshow
    f = f + an(n)*cos(pi*n*t/P) + bn(n)*sin(pi*n*t/P);
end

end
```

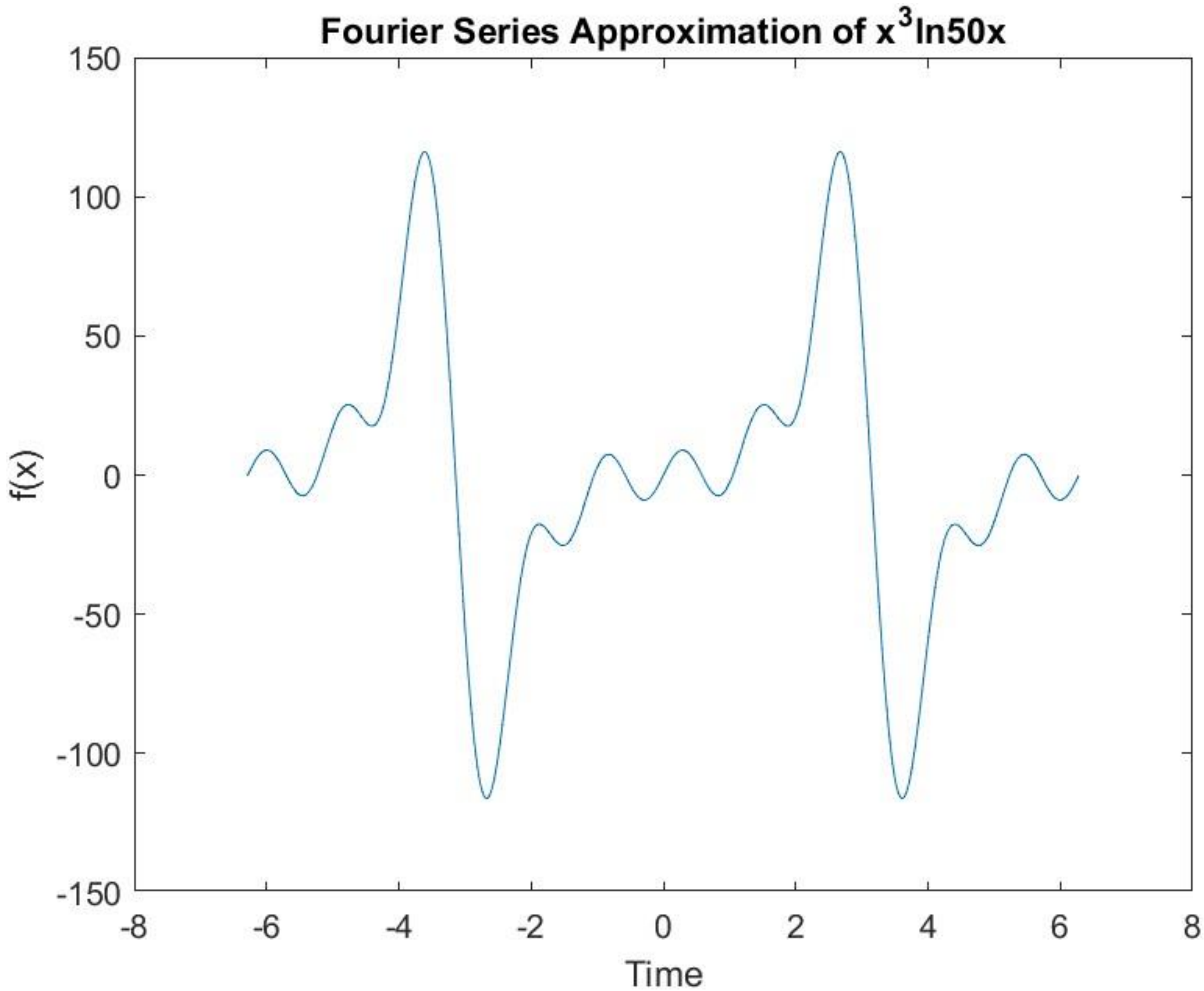
برای مثال یک خروجی از این تابع به ازای این مقادیر می‌گیریم:

```
Num = 10;
P = pi;
a = 50;
b = 3;
Nshow = 5;
[f, t] = FourierSeriesCalculator2(Num, P, a, b , Nshow);
plot(t, f);
xlabel('Time');
ylabel('f(x)');
title('Fourier Series Approximation of x^3ln50x');
```

خروجی ترمینال برای این کد:

```
a0
   0.0000 -24.3523i

an
  -0.0000 +17.6088i
   0.0000 - 7.4022i
  -0.0000 + 3.1417i
   0.0000 - 1.8506i
  -0.0000 + 1.1652i
   0.0000 - 0.8225i
   0.0000 + 0.5993i
  -0.0000 - 0.4626i
   0.0000 + 0.3637i
  -0.0000 - 0.2961i

bn
  36.2091 +12.1567i
 -40.7345 -13.1469i
  30.7333 + 9.6373i
 -23.8274 - 7.4570i
  19.4078 + 6.0505i
 -16.3043 - 5.0804i
  14.0560 + 4.3745i
 -12.3376 - 3.8390i
  10.9947 + 3.4193i
  -9.9104 - 3.0818i
```

نمودار خروجی:



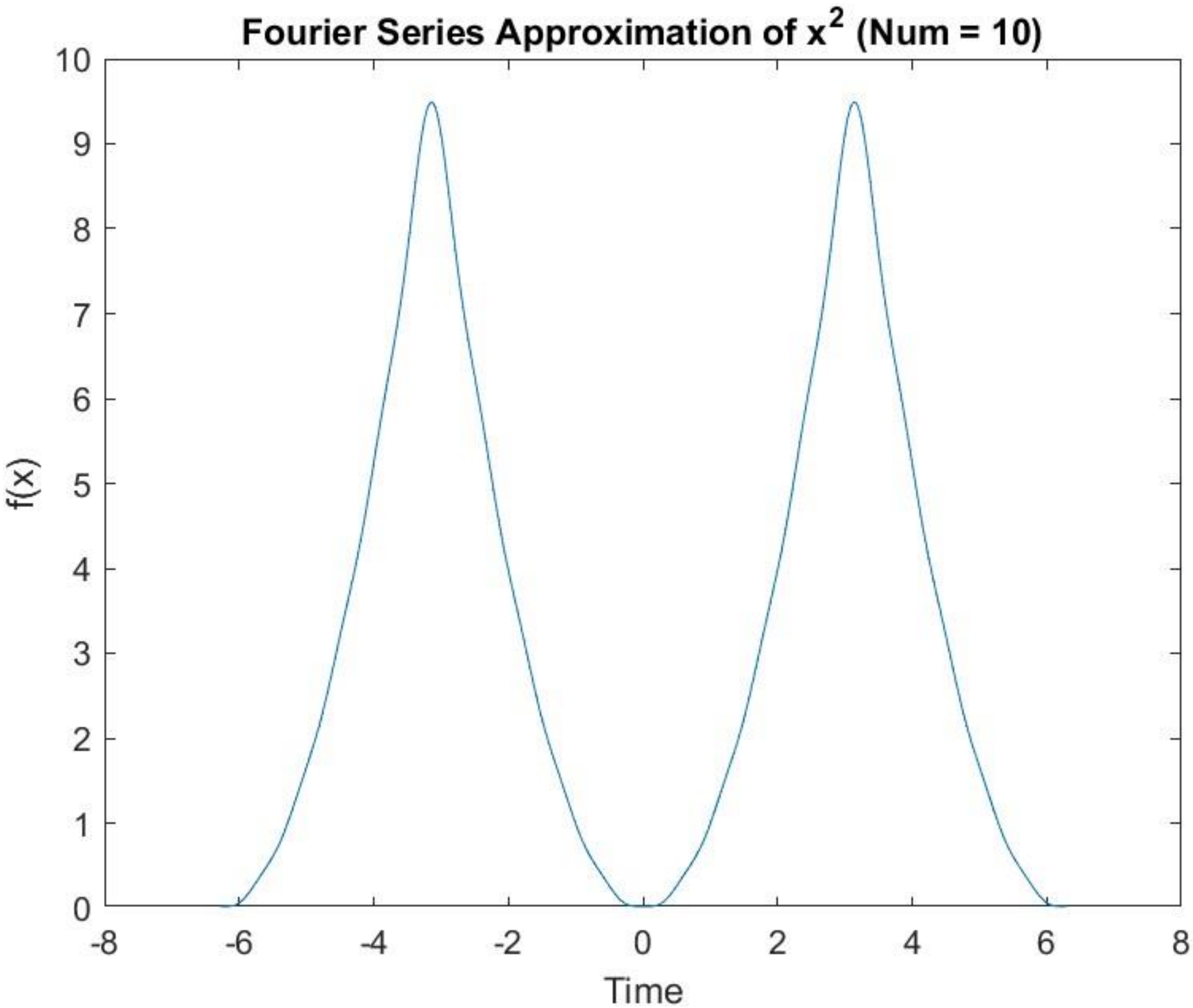
بخش ۳.۲: رسم سری فوریه و مقایسه با تابع اصلی

کد های مربوط به این قسمت: part_2_3_1 و part_2_3_2 و part_2_3_3

از تابع FourierSeriesCalculator1 استفاده می‌کنیم. برای Numهای مختلف مقادیر ترمینال و نمودار کشیده شده را در ادامه می‌بینیم.

برای Num = 10

```
a0
    6.5797

an
   -4.0000
    1.0000
   -0.4444
    0.2500
   -0.1600
    0.1111
   -0.0816
    0.0625
   -0.0494
    0.0400

bn
   1.0e-14 *

   -0.0203
    0.0477
         0
   -0.0159
    0.0424
   -0.1133
    0.1789
   -0.0278
    0.0265
    0.0976
```



برای Num = 50

a0

6.5797

an: -4.0000 1.0000 -0.4444 0.2500 -0.1600 0.1111 -0.0816 0.0625 -0.0494 0.0400 -0.0331

0.0278 -0.0237 0.0204 -0.0178 0.0156 -0.0138 0.0123 -0.0111 0.0100 -0.0091 0.0083

-0.0076 0.0069 -0.0064 0.0059 -0.0055 0.0051 -0.0048 0.0044 -0.0042 0.0039 -0.0037

0.0035 -0.0033 0.0031 -0.0029 0.0028 -0.0026 0.0025 -0.0024 0.0023 -0.0022 0.0021

-0.0020 0.0019 -0.0018 0.0017 -0.0017 0.0016

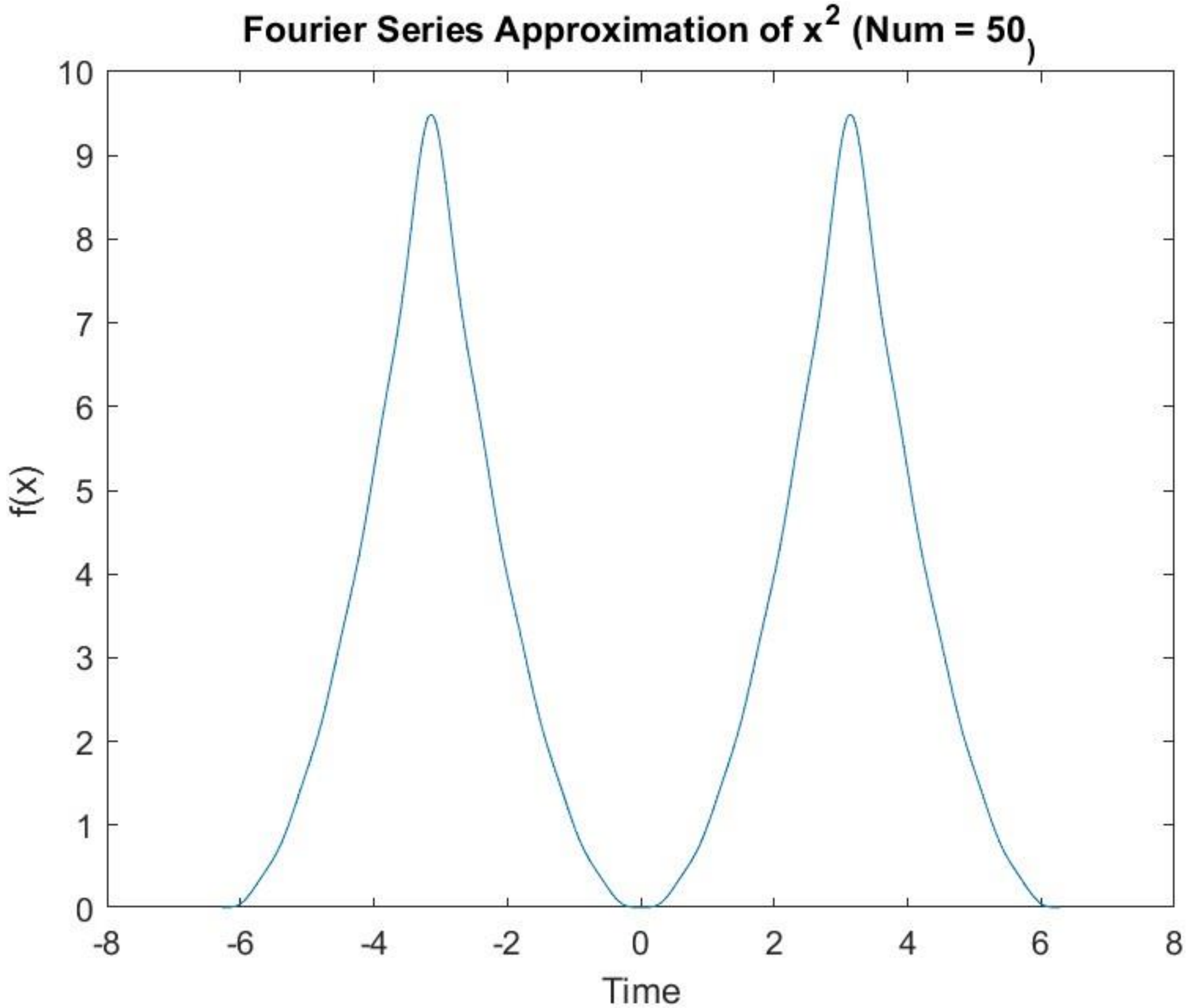
bn: 1.0e-14 * -0.0203 0.0477 0 -0.0159 0.0424 -0.1133 0.1789 -0.0278 0.0265 0.0976

-0.0331 -0.0707 0.1970 -0.1330 -0.0336 0.0292 -0.0795 0.1820 0.0371 -0.0486 0.0141

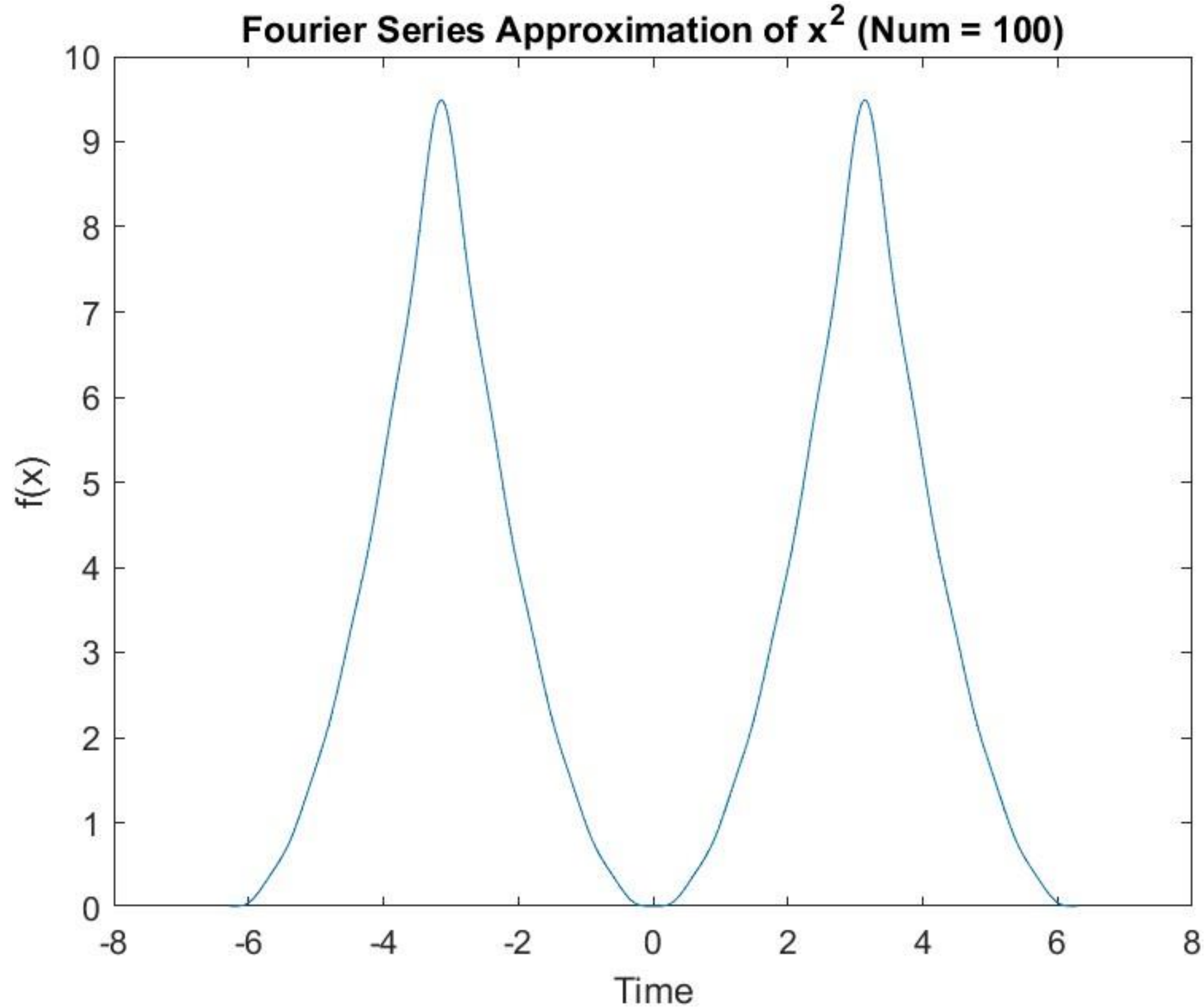
-0.0689 0.0543 0.0070 -0.0398 0.1639 -0.1488 -0.1030 0.2507 -0.2739 0.4087 -0.3194

0.0699 0.2644 -0.4966 0.6416 -0.5724 0.3330 0.1622 -0.3101 0.3545 -0.3072 0.2684

-0.2105 0.1119 0.1181 0.1658 -0.0703 -0.2006 -0.1429



برای Num = 100: چون تعداد اعدا ترمینال زیاد است از نمایش صرف نظر می‌کنیم.



همان طور که می‌دانیم، نمایش سری فوریه در واقع همان تابع را بیان می‌کند. بنابراین با رسم آن باید به شکل تقریبی تابع اصلی برسیم. هرچه تعداد جملاتی که محاسبه می‌کنیم بیشتر باشد، دقت نمودار رسم شده بهتر خواهد بود. زمانی که تعداد جملات کمتری را محاسبه می‌کنیم، ممکن است نمودار خیلی دقیق نباشد و در برخی جاها شاید نوسان باشیم. اما این مشکل را می‌توان با افزایش تعداد جملات حل کرد.

همچنین در خروجی نشان داده شده، ما تابع را به شکل متناوب درآورده‌ایم. به این معنا که در هر دوره تناوب تابع باید به شکل تابع x^2 باشد. که در نمودار ها مشخص است.

بخش ۲.۴: محاسبه حد مجموع و تطبیق نتایج

کد های این قسمت: FourierSeriesCalculator3 و part_2_4

ابتدا به کمک فرمول های سری فوریه، به صورت دستی سری فوریه تابع x^2 را محاسبه می‌کنیم: (از زوج بودن این تابع در محاسبات استفاده شده است)

$$f(x) = a_0 + \sum_{n=1}^{+\infty} a_n \cos(nx) + b_n \sin(nx)$$

$$a_0 = \frac{1}{2\pi}\int_{-\pi}^{\pi} x^2 \, dx = \frac{1}{3}\pi^2$$

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} x^2 \cos(nx)\ dx = \frac{1}{\pi}\frac{2(\pi^2 n^2 - 2)\sin(n\pi) + 4\pi n\cos(n\pi)}{n^3} = \frac{4(-1)^n}{n^2}$$

$$b_n = 0 \quad \forall n \geq 1$$

$$f(x) = \frac{\pi^2}{3} + 4\sum_{n=1}^{\infty}\frac{(-1)^n}{n^2}\cos(nx)$$

حال مقدار x را برابر π قرار می‌دهیم:

$$\pi^2 = \frac{\pi^2}{3} + 4\sum_{n=1}^{\infty}\frac{(-1)^n}{n^2}\cos(n\pi)$$

$$2\frac{\pi^2}{3} = 4\sum_{n=1}^{\infty}\frac{(-1)^n}{n^2}(-1)^n$$

$$\sum_{n=1}^{\infty}\frac{1}{n^2} = \frac{\pi^2}{6}$$

تابعی که در FourierSeriesCalculaor1 بود را با اندکی تغییر برای این قسمت آماده می‌کنیم. تنها تغییری که نیاز است اعمال شود این است که به جای انتخاب یک بازه برای t، یک مقدار برای آن قرار می‌دهیم که برابر π است. در اینجا نیازی به ضرایب سینوسی و کسینوسی سری فوریه نداریم بنابراین آنها را چاپ نمی‌کنیم و فقط مقدار تابع که همان f است و DC چاپ می‌شود:

```
function [f, t] = FourierSeriesCalculator3(Num, P, a, Nshow)
t = pi;
a_0 = (1/(P)) * integral(@(x) x.^a, -P, P);
a_n = zeros(Num, 1);
b_n = zeros(Num, 1);
for n = 1:Num
    a_n(n) = (1/P) * integral(@(x) x.^a .* cos((pi*n*x)/P), -P, P);
    b_n(n) = (1/P) * integral(@(x) x.^a .* sin((pi*n*x)/P), -P, P);
end
f = (a_0/2);
for n = 1:Nshow
    f = f + a_n(n)*cos(pi*n*t/P) + b_n(n)*sin(pi*n*t/P);
end
disp('f')
disp(f)
disp('a0')
disp(a0)
end
```

پس از اجرای کد زیر:

```
Num = 100;
P = pi;
a = 2;
Nshow = 100;
[f, t] = FourierSeriesCalculator3(Num, P, a, Nshow);
```

خروجی برای t=π در ترمینال نمایش داده خواهد شد که برابر ۹.۸۲۹۸ می‌باشد. مقدار$a_0/2$ برابر ۳.۲۸۹۹ خواهد بود.

اگر به صورت دستی این حاصل را حساب کنیم داریم:

$$9.8298 = 3.289 + 4\sum_{n=1}^{\infty}\frac{(-1)^n}{n^2}(-1)^n$$

$$\sum_{n=1}^{\infty}\frac{1}{n^2} = \frac{9.8298 - 3.289}{4} = 1.6352$$

$$\sum_{n=1}^{\infty}\frac{1}{n^2} = \frac{\pi^2}{6} = 1.6449$$

میزان خطا برابر ۰.۰۰۹۷ است که خطای کمی است.

بخش ۲.۵: آنالیز هارمونیک در سری فوریه

کد های این قسمت: HarmonicFourierSeries و part_2_5

برای نوشتن تابع برای آنالیز هارمونیک، از همان فرمول‌های دستور پروژه استفاده می‌کنیم و آن ها را به صورت کد متلب می‌نویسیم. کد به شکل زیر خواهد بود:

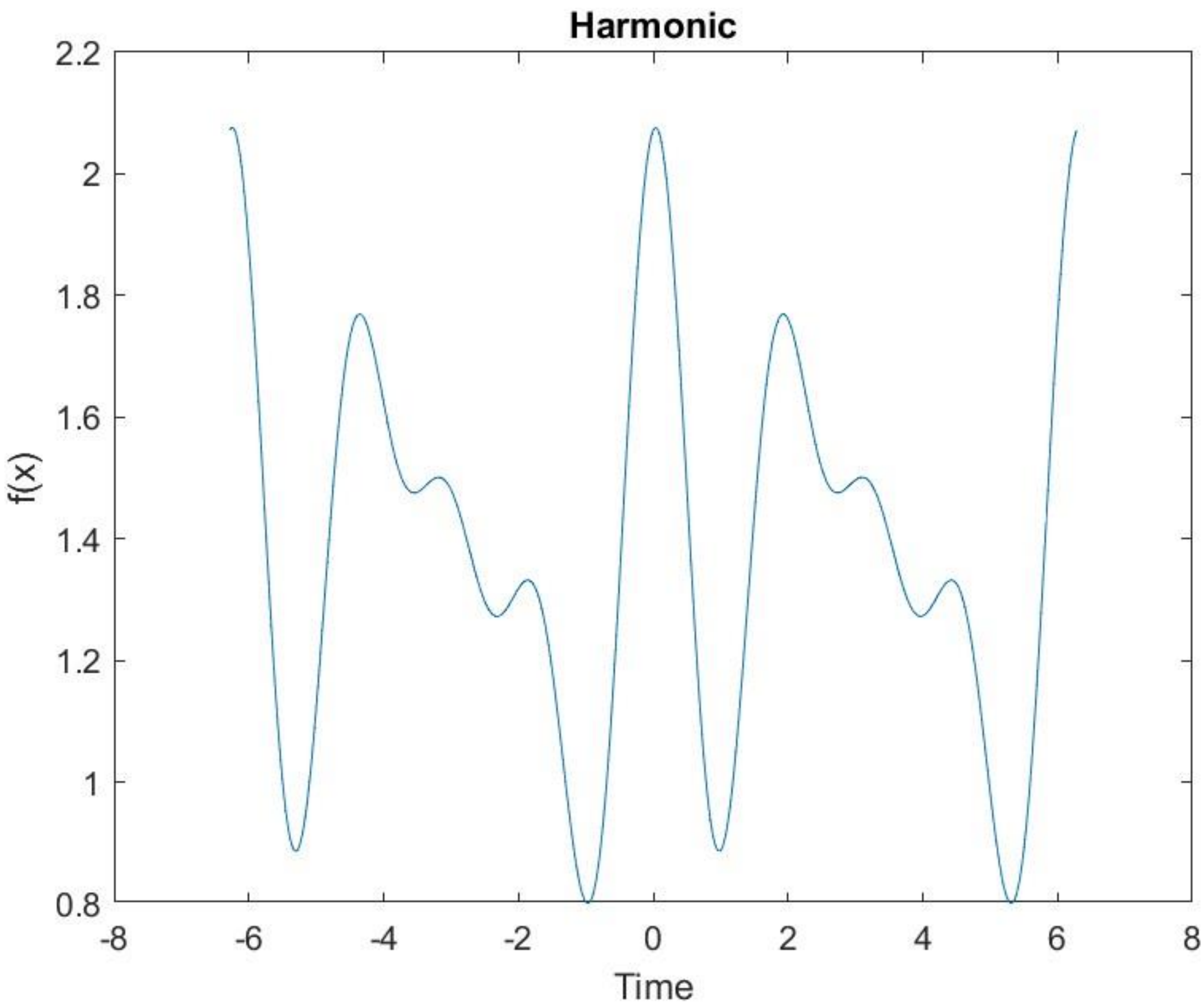
```
function [f, t] = HarmonicFourierSeries(x, fx, Nshow, P)
t = linspace(-2*P, 2*P, 1000);
num_points = numel(x);
a0 = 2 * mean(fx);
disp('a0')
disp(a0)
an = zeros(Nshow, 1);
bn = zeros(Nshow, 1);

for n = 1:Nshow
    sum_cos = 0;
    sum_sin = 0;
    for i = 1:num_points
    sum_cos = sum_cos + fx(i) * cos(n * x(i));
    end
    mean_cos = sum_cos / num_points;
    an(n) = 2 * mean_cos;

    for i = 1:num_points
    sum_sin = sum_sin + fx(i) * sin(n * x(i));
    end
    mean_sin = sum_sin / num_points;
    bn(n) = 2 * mean_sin;
end

f = (a0/2);
for n = 1:Nshow
    f = f + an(n)*cos(n*t) + bn(n)*sin(n*t);
end
```

برای اجرای تابع:

```
x = [0, pi/3, (2*pi)/3, pi, (4*pi)/3, (5*pi)/3, 2*pi];
fx = [1, 1.4, 1.9, 1.7, 1.5, 1.2, 1];
P = pi;
Nshow = 4;

[f, t] = HarmonicFourierSeries(x, fx, Nshow, P);

plot(t, f);
xlabel('Time');
ylabel('f(x)');
title('Harmonic');
```

نقاط داده شده در دستور را به تابع می‌دهیم تا نمودار را رسم کند.

نمودار:



اگر ضرایب را در تابعی که نوشتیم چاپ کنیم داریم: توجه: با توجه به هماهنگی صورت گرفته با چیف درس، ضرایب نوشته شده در صورت پروژه اشتباه است

a0 2.7714

an -0.0286 0.2000 0.3143 0.2000 **bn** 0.1485 -0.0495 -0.0000 0.0495

بخش ۳: تبدیل فوریه

بخش ۳.۲: بررسی حوزه زمان و فرکانس چند تابع

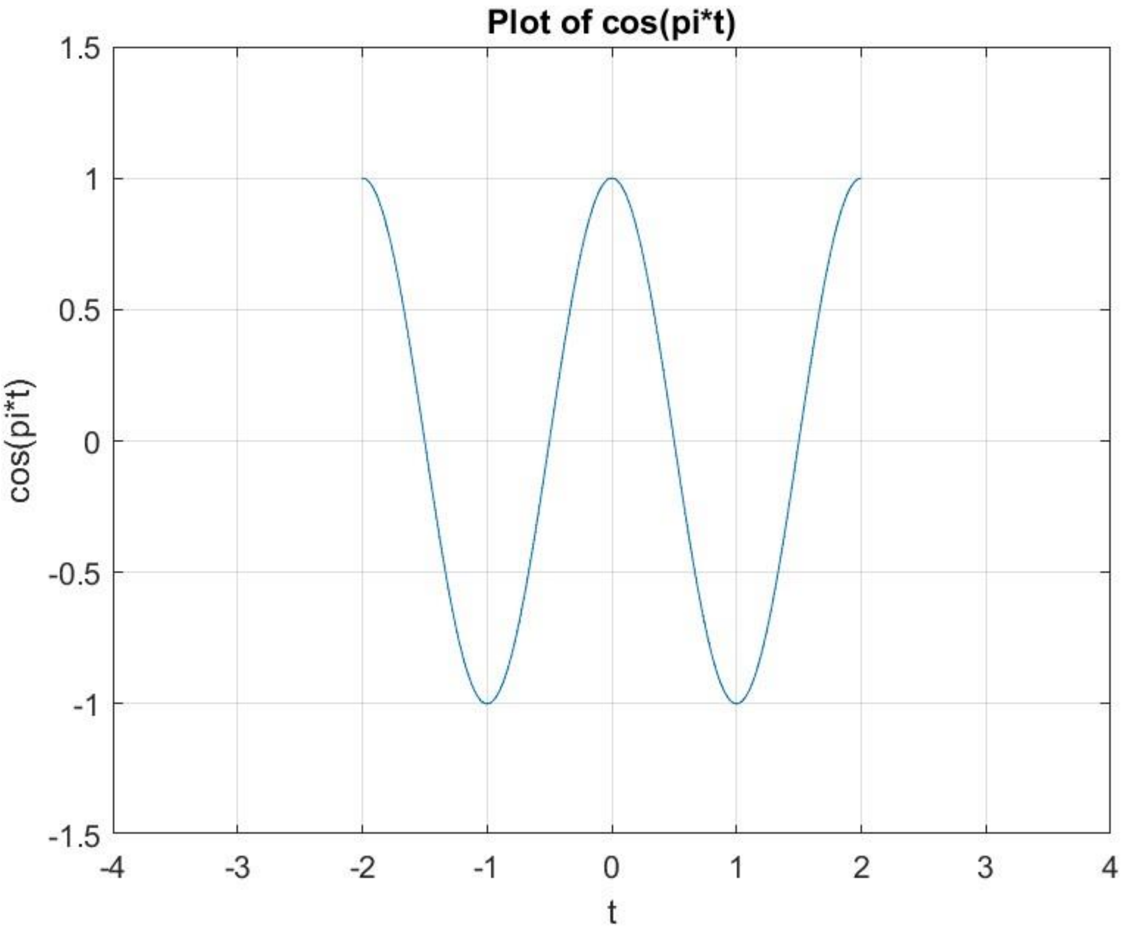
کد های این قسمت: part3_2_1_1 part_3_2_1_2 part_3_2_1_3 part_3_2_2_1 part_3_2_2_2 part_3_2_2_3 part_3_2_3_1 part_3_2_3_2 part_3_2_3_3

تابع $\cos(\pi t)$:

به کمک قطعه کد زیر تابع را رسم می‌کنیم:

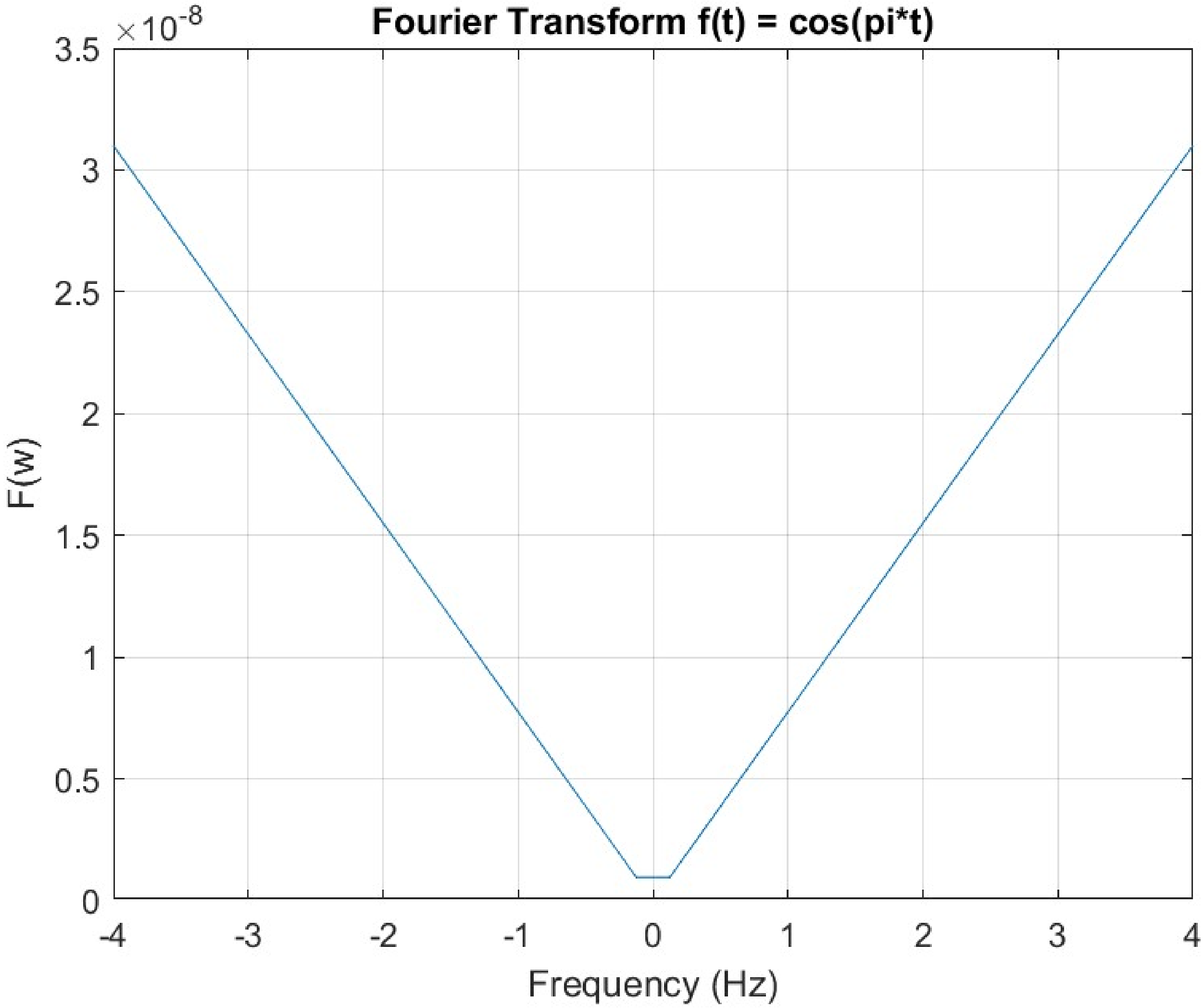
```
fs = 100000;
t = -2:1/fs:2;
y = cos(pi*t);

plot(t,y)
xlim([-4 4])
ylim([-1.5 1.5])
xlabel('t')
ylabel('cos(pi*t)')
title('Plot of cos(pi*t)')
grid on
```

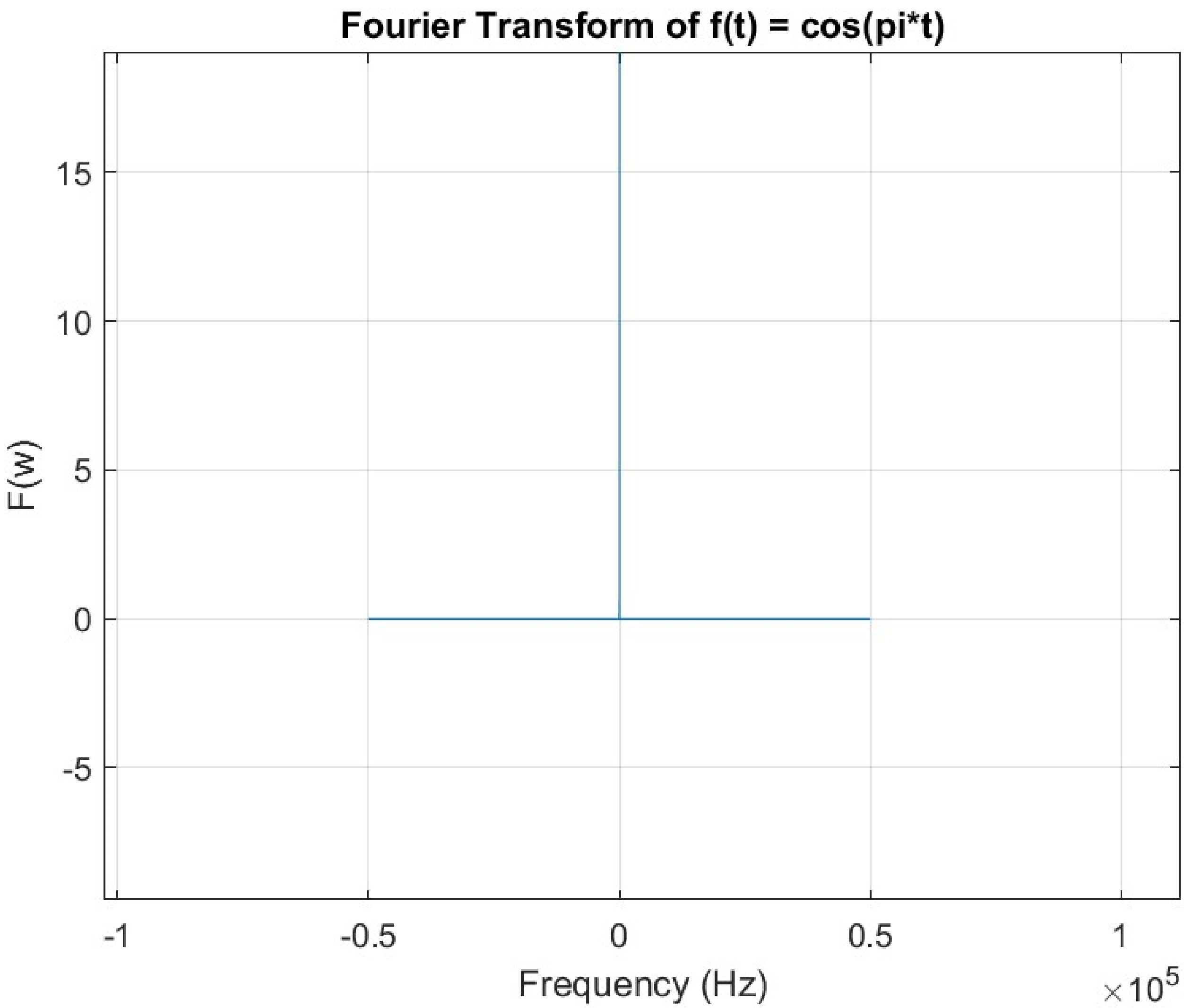
نمودار خروجی:



سپس به کمک fft و fftshift تبدیل فوریه را محاسبه کرده و رسم می‌کنیم:

نمودار:



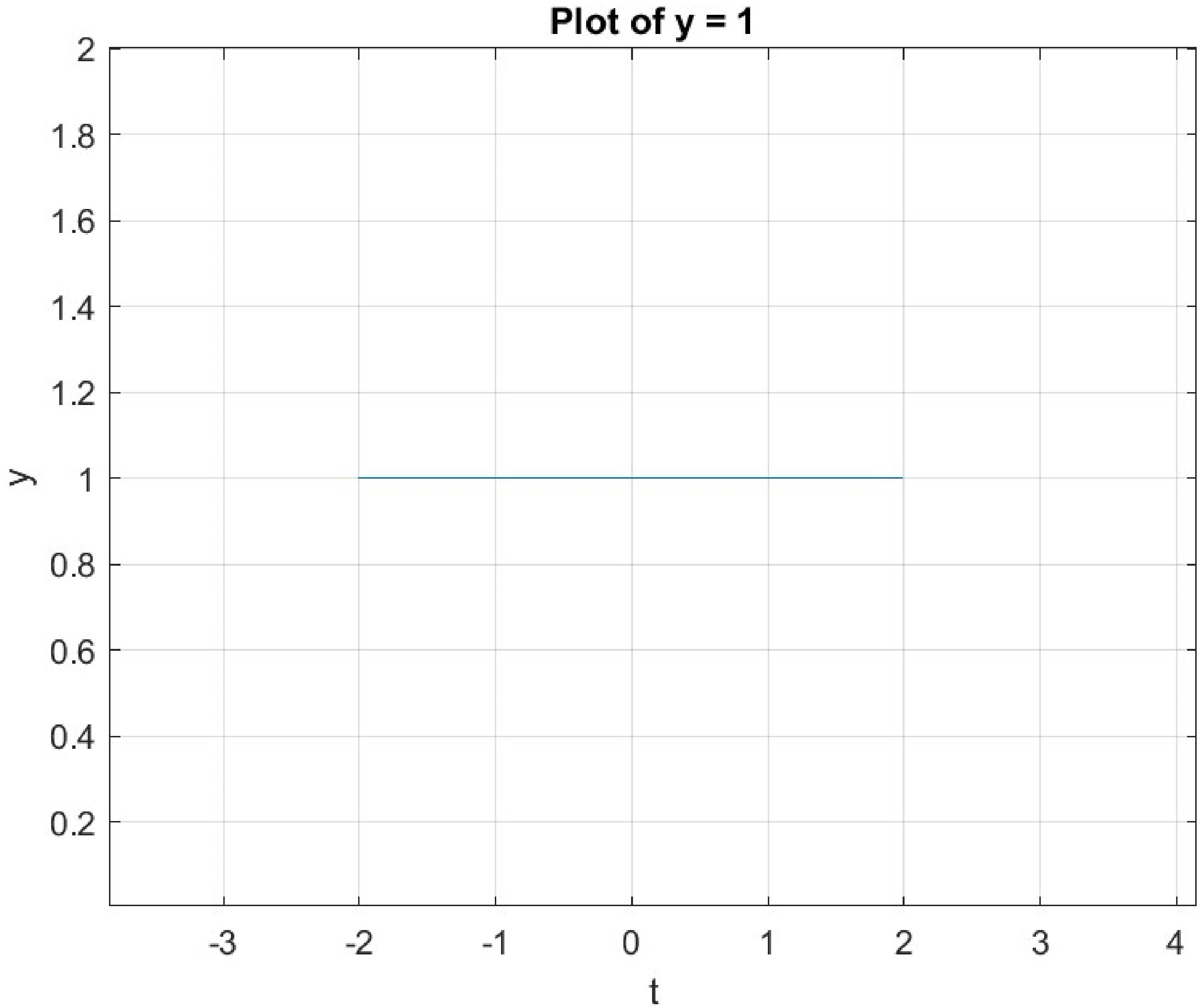


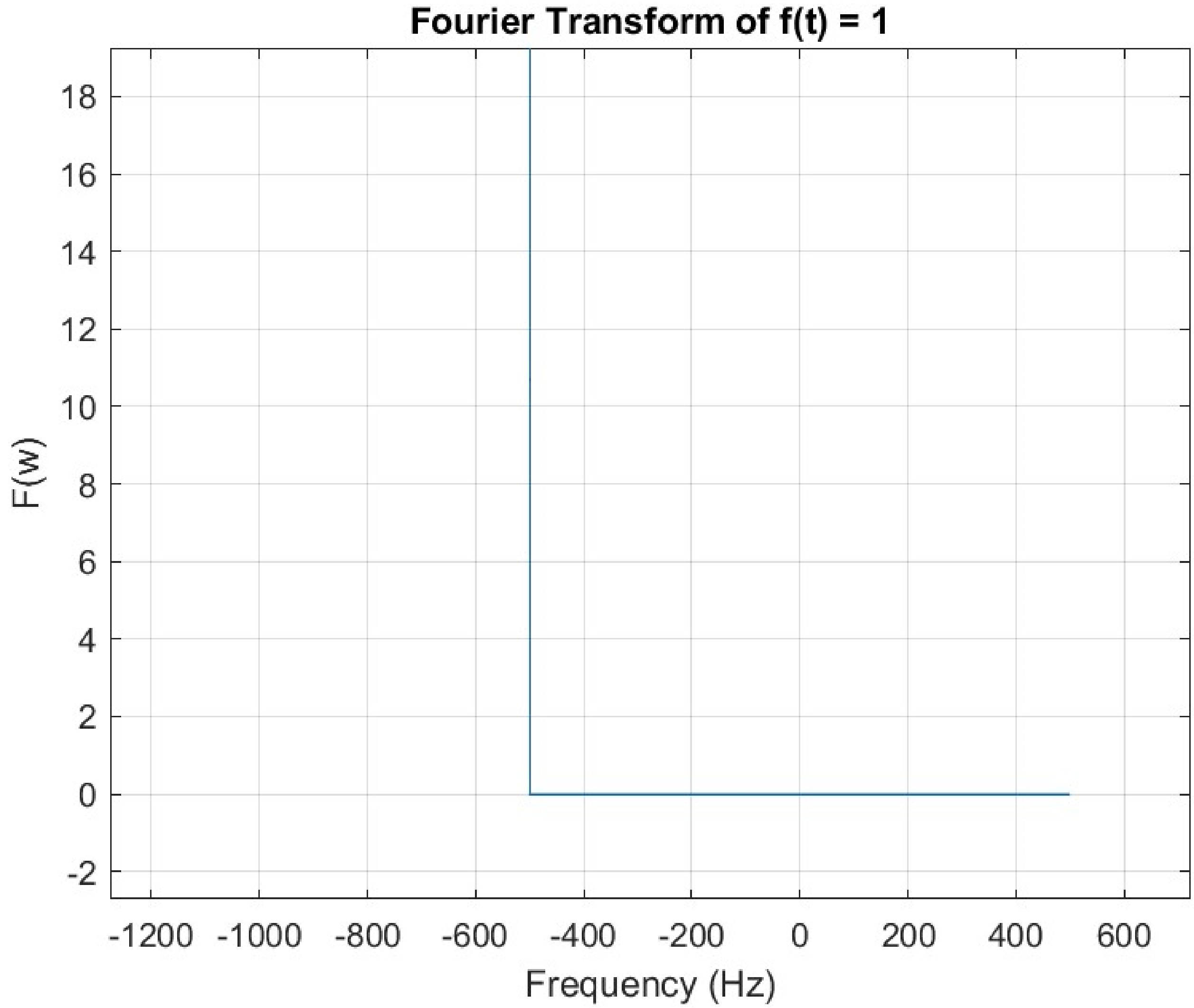
محاسبه دستی:

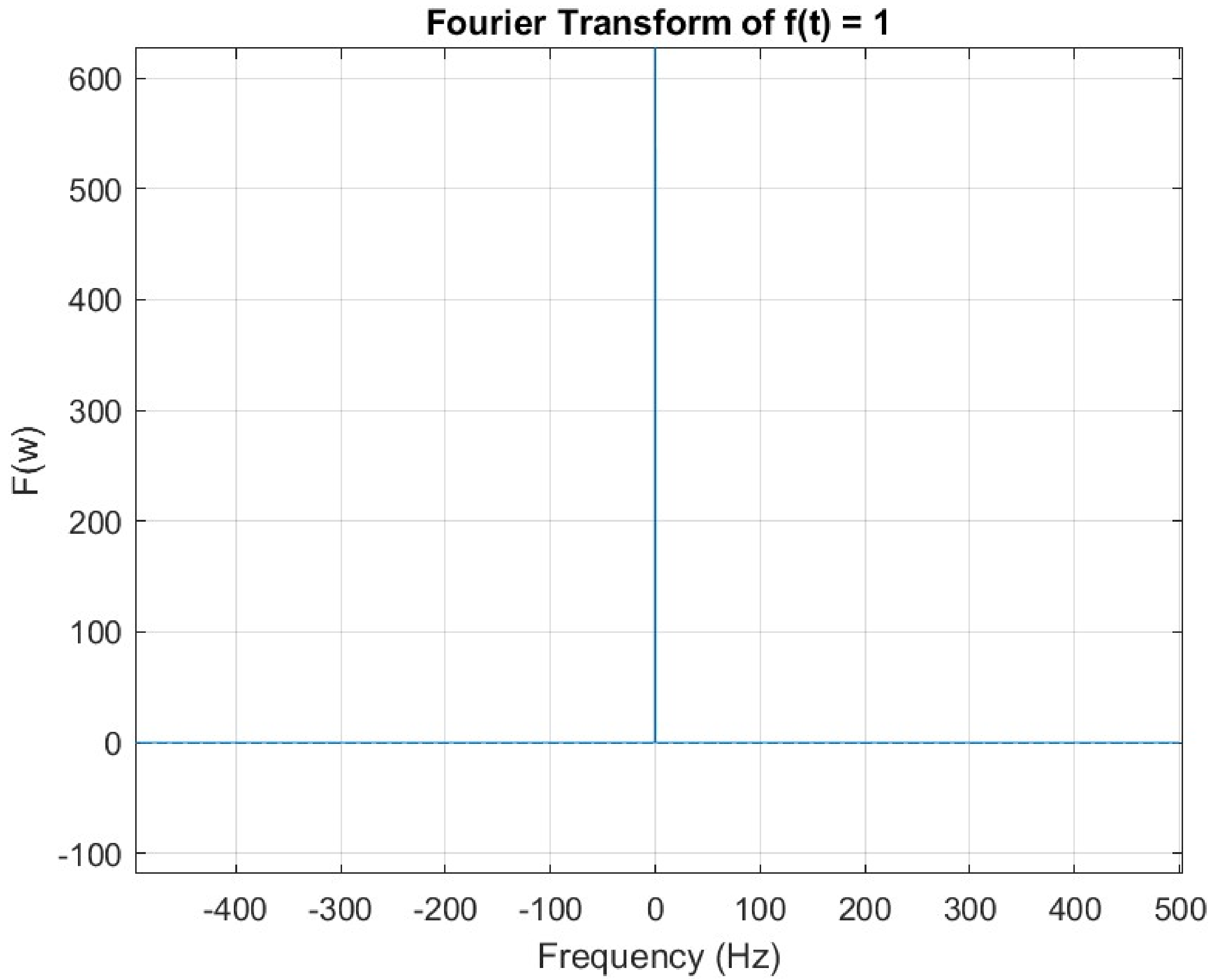
$$f(t) = \cos(\omega_0 t) \rightarrow F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt = \int_{-\infty}^{+\infty} \cos(\omega_0 t) e^{-i\omega t} dt$$

$$F(\omega) = \pi\delta(\omega - \omega_0) + \pi\delta(\omega + \omega_0)$$

تابع $F(x) = 1$




Fourier Transform of f(t) = 1
F(w)
Frequency (Hz)
18
16
14
12
10
8
6
4
2
0
-2
-1200
-1000
-800
-600
-400
-200
0
200
400
600

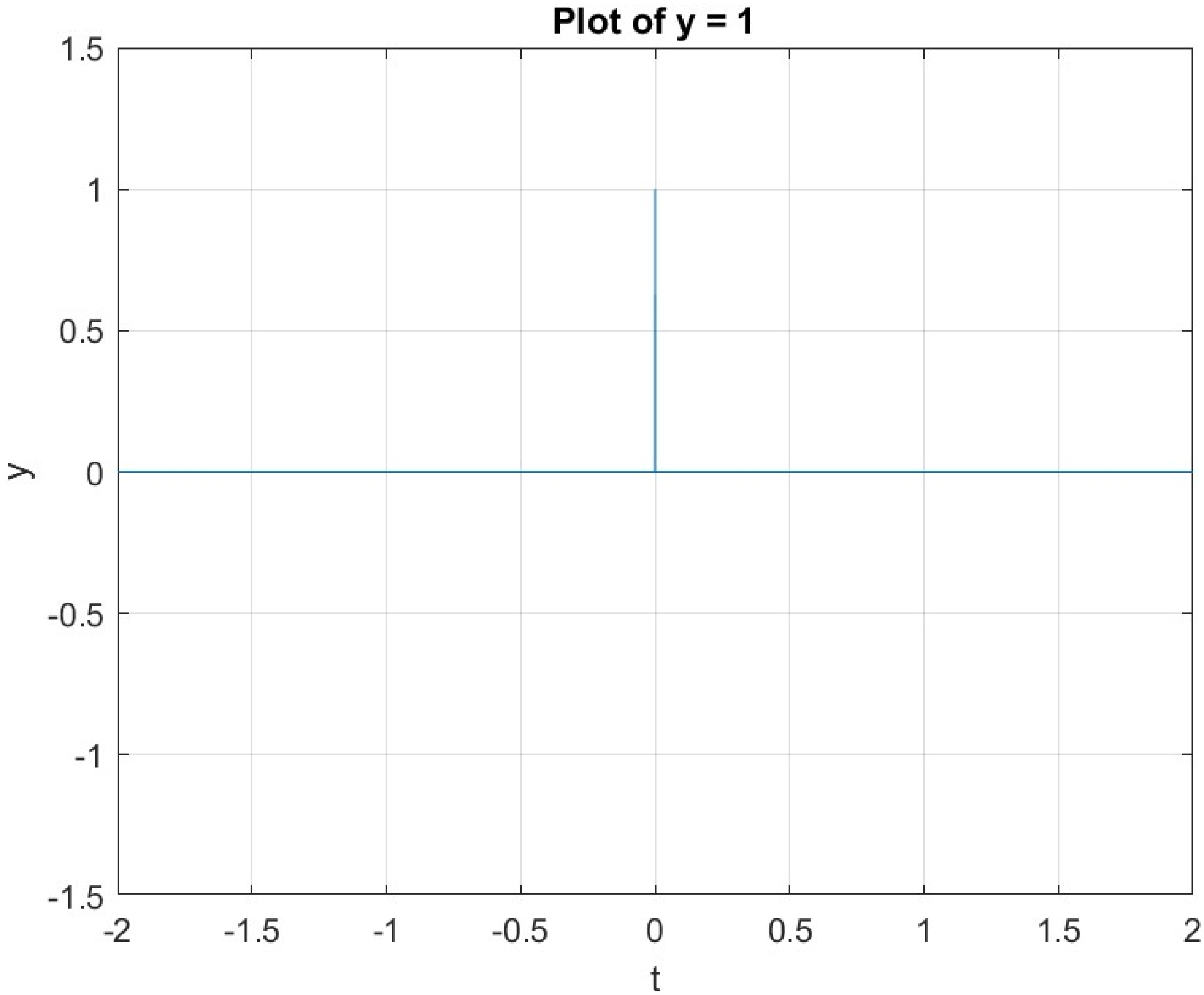


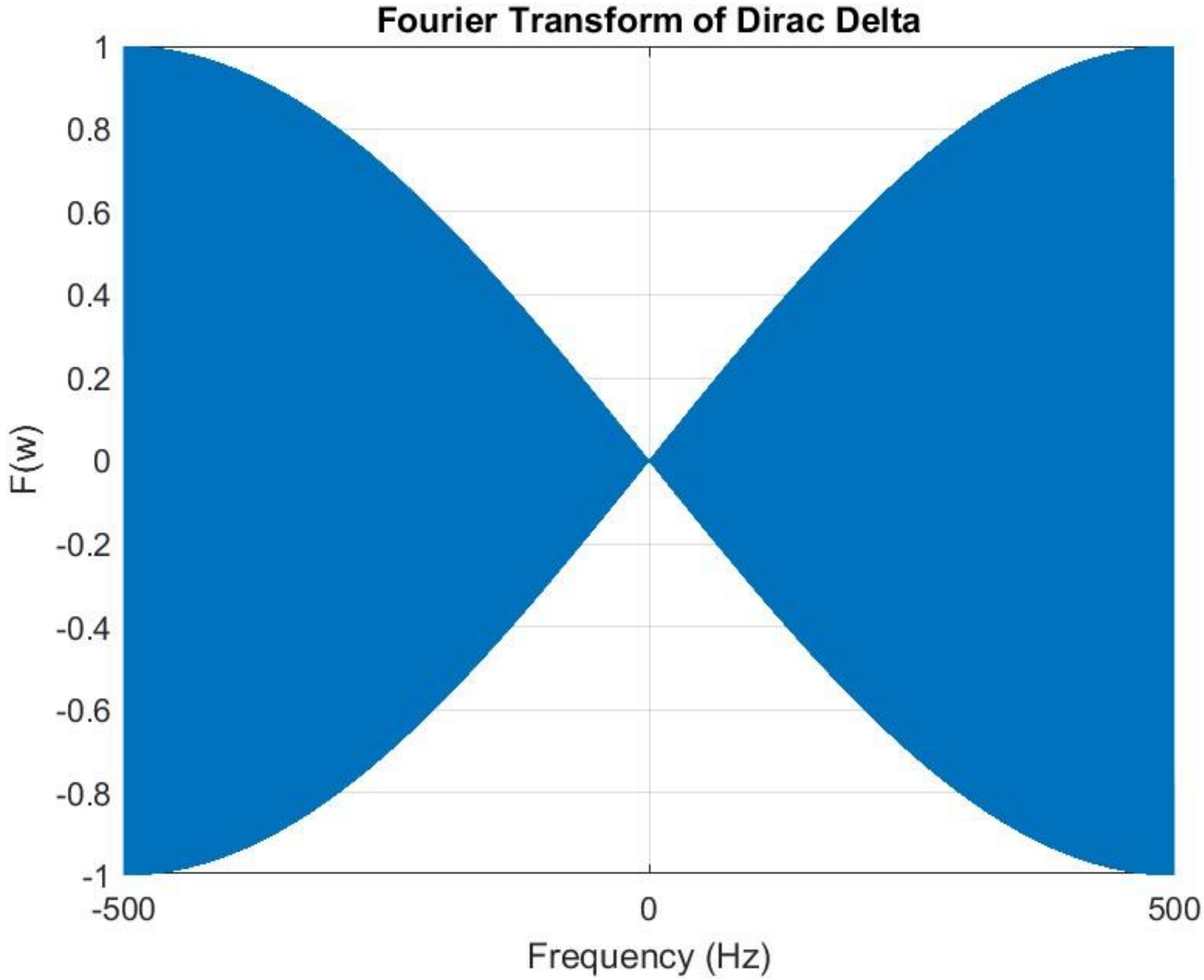
محاسبه دستی:

$$f(t) = 1 \ \rightarrow F(\omega) = \int_{-\infty}^{+\infty} 1 \, . \, e^{-i\omega t} \, dt$$

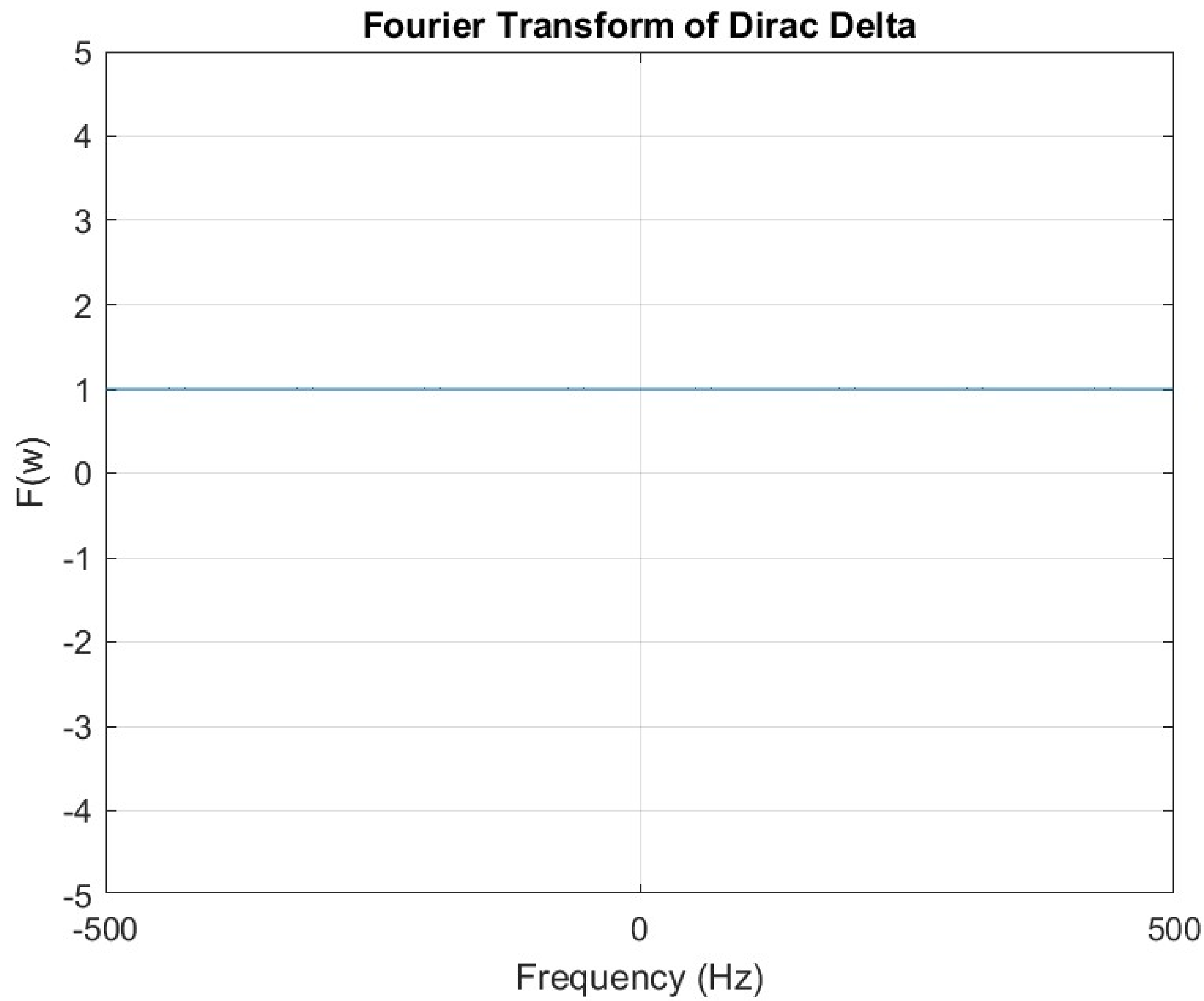
$$F(\omega) = 2\pi\delta(\omega)$$

$f(x) = \delta(x)$تابع





Fourier Transform of Dirac Delta
F(w)
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-500
0
500
Frequency (Hz)



محاسبه دستی:

$$f(t) = \delta(t) \rightarrow F(\omega) = \int_{-\infty}^{+\infty} \delta(t) e^{-i\omega t} dt$$

$$F(\omega) = 1$$

همان طور که مشخص است خروجی ها با محاسبه های دستی تطابق دارند.

در واقع خروجی هایی که بعد از اعمال fftshift رسم کردیم درست هستند. در واقع تابع fft تبدیل فوریه تابع داده شده را محاسبه می‌کند اما محور مربوط به فرکانس صحیح نیست و باید شیفت بخورد. در واقع خروجی تابع fft به شکل zero centered نیست و برای نمایش بصری بهتر باید fftshift اعمال شود.

بخش ۳.۳: موسیقی

کد های این قسمت: part_3_3_1 part_3_3_2 part_3_3_3 part_3_3_4 part_3_3_5 part_3_3_6

برای تغییر فرکانس یک فایل صوتی در متلب ابتدا آن را به کمک تابع audioread باز می‌کنیم. خروجی این تابع در واقع شامل فرکانس هم هست. اگر فرکانس را برای فایل اصلی چاپ کنیم: (Fs): 44100

همچنین اگر خروجی های تابع audioread را به تابع sound بدهیم، در محیط متلب فایل صوتی پخش می‌شود.

حال برای تغییر فرکانس ابتدا فایل اصلی را باز می‌کنیم، فرکانس را n برابر می‌کنیم و از y که خروجی تابع audioread است مجددا با فرکانس جدید نمونه برداری می‌کنیم و در نهایت به کمک audiowrite در یک فایل جدید ذخیره می‌کنیم.

می‌توان فرکانس فایل های جدید را هم مثل فایل اصلی چاپ کرد که دقیقا دو برابر یا نصف فایل اصلی هستند.

فرکانس این فایل همان طور که اشاره شد ۴۴۱۰۰ هرتز می‌باشد. در واقع این فرکانس، فرکانس استاندارد برای فایل های صوتی دیجیتال است. یکی از دلایل انتخاب این فرکانس قضیه نایکوئیست است. قضیه نایکوئیست بیان می‌کند که برای گرفتن دقیق سیگنال پیوسته، نرخ نمونه‌برداری باید حداقل دو برابر بالاترین فرکانس موجود در سیگنال باشد. شنوایی انسان معمولاً بین ۲۰ هرتز تا ۲۰ کیلوهرتز است. بنابراین، نرخ نمونه برداری حداقل ۴۰ کیلوهرتز برای گرفتن تمام فرکانس های شنیداری مورد نیاز است.

با تغییر فرکانس این فایل صوتی تغییراتی رخ می‌دهد که یکی از آنها تغییر حجم فایل های صوتی است. با تغییر فرکانس حجم فایل های صوتی بیشتر می‌شود. با دو برابر کردن فرکانس برخی قسمت های موسیقی کمی ناواضحی دارد. همچنین در برخی قسمت ها کشیدگی کلمات احساس می‌شود.

با کاهش فرکانس، بدون تغییر دادن میزان ولوم دستگاه، موسیقی بلندتر شنیده می‌شود. قسمت هایی که شنیده می‌شود به واضحی فایل اصلی نیست. همچنین در قسمت های پایانی مقدار قطعی داریم. قسمت گیتار انتهایی به شدت بلند احساس می‌شود.