Digital Signal Processing: Assignment #1

Due on Monday, March 17, 2014

Prof. Lee

Kevin Zhao, E14982022

Contents

Problem 1 **3**
- a. Square Wave 3
- b. Sawtooth Wave 3
- c. Triangular Wave 3

Problem 2 **4**
- Square Wave 4
- Sawtooth Wave 5
- Triangular Wave 6
- Code 7

Problem 3 **10**

Problem 1

a. Square Wave

Fourier Expansion of Square Wave (complex form)

$$\begin{aligned}
x(t) &= \sum_{k=-\infty}^{\infty} c_k e^{k2\pi f_0 t} \\
&= c_0 + \frac{2}{\pi}\sum_{k=1}^{\infty}[\frac{1}{j\pi(2k-1)}e^{j(2k-1)\omega_0 t} + \frac{1}{-j\pi(2k-1)}e^{-j(2k-1)\omega_0 t}] \\
&= \frac{1}{2} + \frac{2}{\pi}\sum_{k=1}^{\infty}\frac{sin((2k-1)\omega_0 t)}{2k-1}
\end{aligned}$$

b. Sawtooth Wave

Fourier Expansion of Sawtooth Wave

$$x(t) = \frac{1}{2} - \frac{1}{\pi}\sum_{k=1}^{\infty}\frac{1}{k}sin(k\omega_0 t)$$

c. Triangular Wave

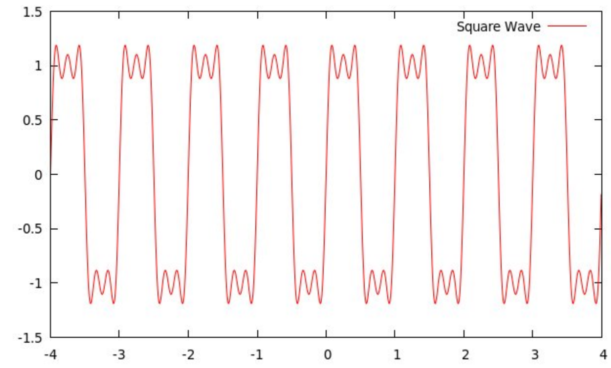
Fourier Expansion of Triangular Wave

$$x(t) = \frac{1}{2} + \frac{4}{\pi^2}\sum_{k=1}^{\infty}\frac{sin((k\pi/2)\omega_0 t)}{k^2}sin(2k\pi\omega_0 t)$$

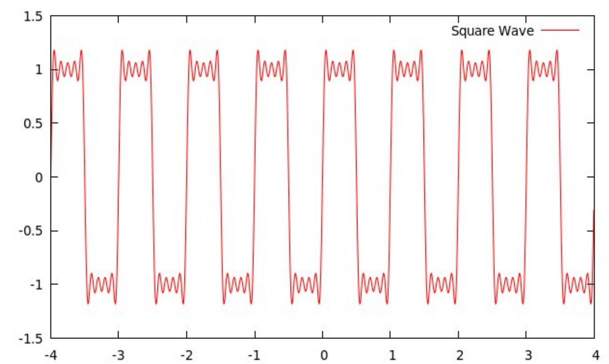
Problem 2

Square Wave

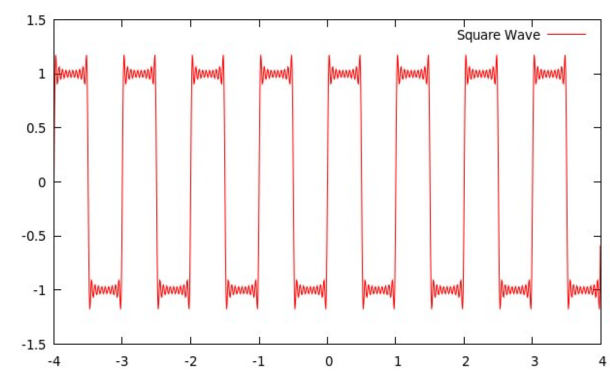
N = 3



N = 5



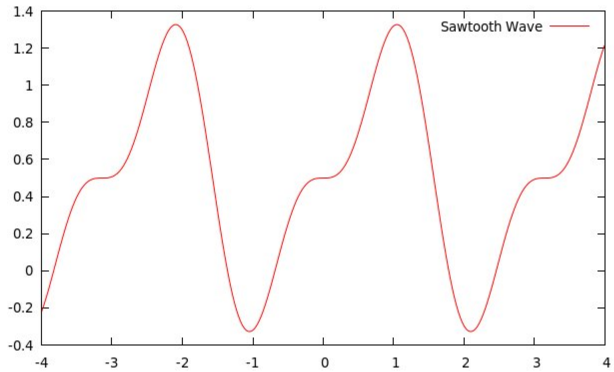
N = 10



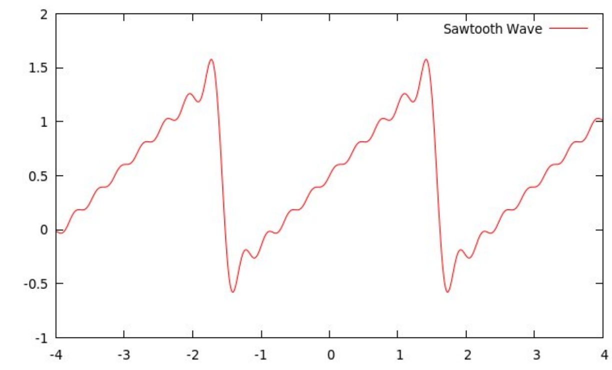
Gibb's phenomenon get significantly when N grows larger. In the steady state (N is greater than 30), the phenomenon exists with about 9% error.

Sawtooth Wave

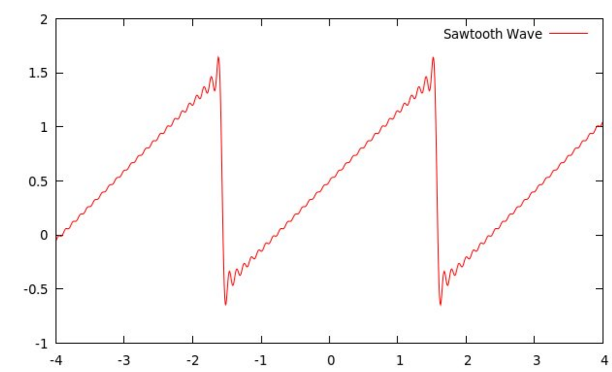
N = 3



N = 10



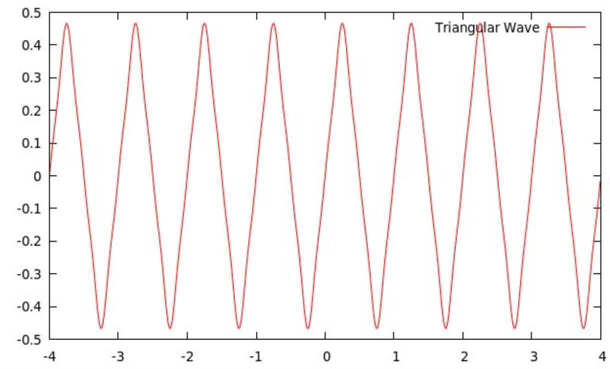
N = 30



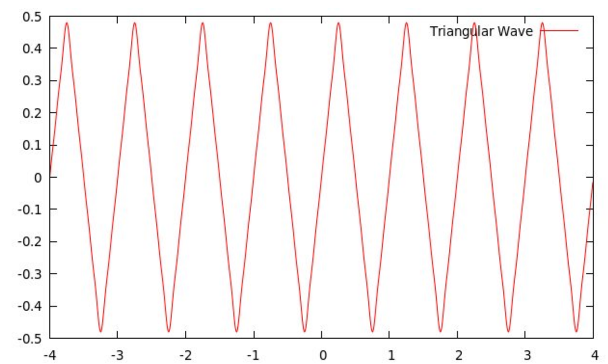
In triangular wave, the oscillation is more obvious. Since that the ramp signal is not easy to follow than square wave.

Triangular Wave

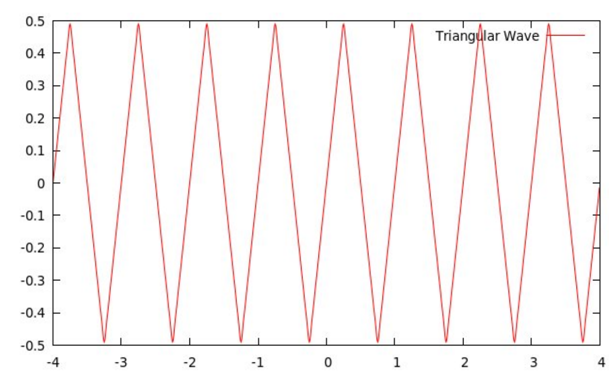
N = 3



N = 5



N = 10



In comparison, triangular wave is more smooth and get converge when N =10. Because the slop deviation not significant as ramp or square.

Code

```
/* Start reading here */
#include <fftw3.h>
#include "gnuplot_i.h"
#define NUM_POINTS 1024

/* Never mind this bit */
#include <stdio.h>
#include <math.h>

unsigned int EXPAN_TERM = 10;
unsigned int SLEEP_LGTH  = 1;

double time_series     [NUM_POINTS];
double square_wave     [NUM_POINTS];
double triangle_wave   [NUM_POINTS];
double sawtooth_wave   [NUM_POINTS];

void init_time_series(double t_range[2]) {
    double time_length = t_range[1] - t_range[0];
    double time_step = time_length / NUM_POINTS;
    double t;
    int i;
    for (i =0, t = t_range[0]; i< NUM_POINTS; ++i, t += time_step) {
        time_series[i] = t;
    }
}

void init_square_wave(double t_range[2],
                      double period,
                      double ts[NUM_POINTS],
                      double f[NUM_POINTS])
{
    double time_length = t_range[1] - t_range[0];
    double time_step = time_length / NUM_POINTS;
    double t;
    int i, A = 1;
    int pn = period / time_step, p2 = pn >> 2;

    for (i =0, t = t_range[0]; i< NUM_POINTS; i++, t += time_step) {
        if ((i%pn) > p2)
            f[i] = A;
        else
            f[i] = -A;
        ts[i] = t;
    }
}
/* Resume reading here */
#define SQUARE_WAVE 1
#define TRIANGLE_WAVE 2
#define SAWTOOTH_WAVE 3
```

```c
void fourier_expansion(unsigned int sig_type) {
    switch (sig_type) {
        case SQUARE_WAVE:
        {
            // i: time index, k: expansion term
            int i, k;
            // linear combination of sinusoids
            double lincom =0;
            /* Whole time series */
            for (i =0; i < NUM_POINTS; ++i) {
                /* Expansion term */
                double t = time_series[i];
                for (k =0; k < EXPAN_TERM; ++k) {
                    lincom += sin(2*M_PI*(2*k+1) * t) / (2*k+1);
                }
                square_wave[i] = 4*lincom/M_PI;
                // reset the accumulation term
                lincom =0;
            }
        }
        break;

        case TRIANGLE_WAVE:
        {
            // i: time index, k: expansion term
            int i, k;
            // linear combination of sinusoids
            double lincom =0;
            /* Whole time series */
            for (i =0; i < NUM_POINTS; ++i) {
                /* Expansion term */
                double t = time_series[i];
                for (k =0; k < EXPAN_TERM; ++k) {
                    lincom += pow(-1.0,k) * sin(2*M_PI*(2*k+1) * t) /(2*k+1)/(2*k+1);
                }
                triangle_wave[i] = 4*lincom /(M_PI*M_PI);
                // reset the accumulation term
                lincom =0;
            }
        }
        break;

        case SAWTOOTH_WAVE:
        {
            // i: time index, k: expansion term
            int i, k;
            // linear combination of sinusoids
            double lincom =0;
            /* Whole time series */
            for (i =0; i < NUM_POINTS; ++i) {
                /* Expansion term */
                double t = time_series[i];
                for (k =1; k < EXPAN_TERM; ++k) {
```

```
                    lincom += pow(-1.0,k+1) * sin(2*(k*t))/(k);
                }
                sawtooth_wave[i] = 0.5 + 2*lincom /(M_PI);
                // reset the accumulation term
                lincom =0;
            }
        }
    }
}

int main(int argc, char *argv[]) {
    gnuplot_ctrl *h;
    /* Initialize the gnuplot handle */
    printf("*** DSP example of FFT gnuplot through C ***\n") ;
    h = gnuplot_init() ;

    if (argc >1) {
        EXPAN_TERM = atoi(argv[1]);
    } else if(argc ==3) {
        SLEEP_LGTH = atoi(argv[2]);
    }

    /* Initialize time series */
    double time_range[2] = {-4.0,4.0};
    init_time_series(time_range);

    // Fig.1
    fourier_expansion(SQUARE_WAVE);
    gnuplot_resetplot(h) ; gnuplot_setstyle(h, "lines") ;
    printf("\n\n*** Square Wave \n") ;
    gnuplot_plot_xy(h, time_series, square_wave, NUM_POINTS, "Square Wave") ;
    sleep(SLEEP_LGTH) ;

    // Fig.2
    fourier_expansion(TRIANGLE_WAVE);
    gnuplot_resetplot(h) ; gnuplot_setstyle(h, "lines") ;
    printf("\n\n*** Triangular Wave \n") ;
    gnuplot_plot_xy(h, time_series, triangle_wave, NUM_POINTS, "Triangular Wave") ;
    sleep(SLEEP_LGTH) ;

    // Fig.3
    fourier_expansion(SAWTOOTH_WAVE);
    gnuplot_resetplot(h) ; gnuplot_setstyle(h, "lines") ;
    printf("\n\n*** Sawtooth Wave \n") ;
    gnuplot_plot_xy(h, time_series, sawtooth_wave, NUM_POINTS, "Sawtooth Wave") ;
    sleep(SLEEP_LGTH) ;

    printf("\n\n*** end of DSP example ***\n") ;
    gnuplot_close(h);
    return 0;
}
```

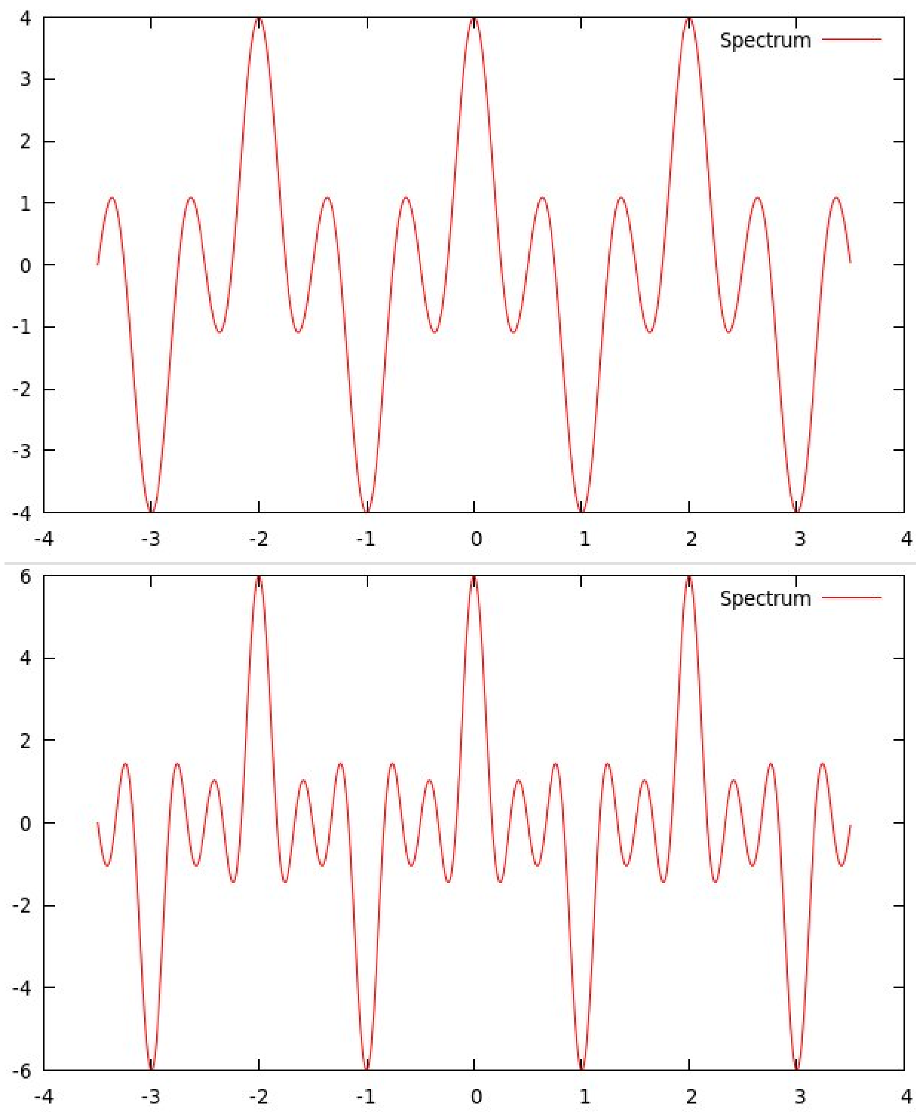
Problem 3

The transfer function is

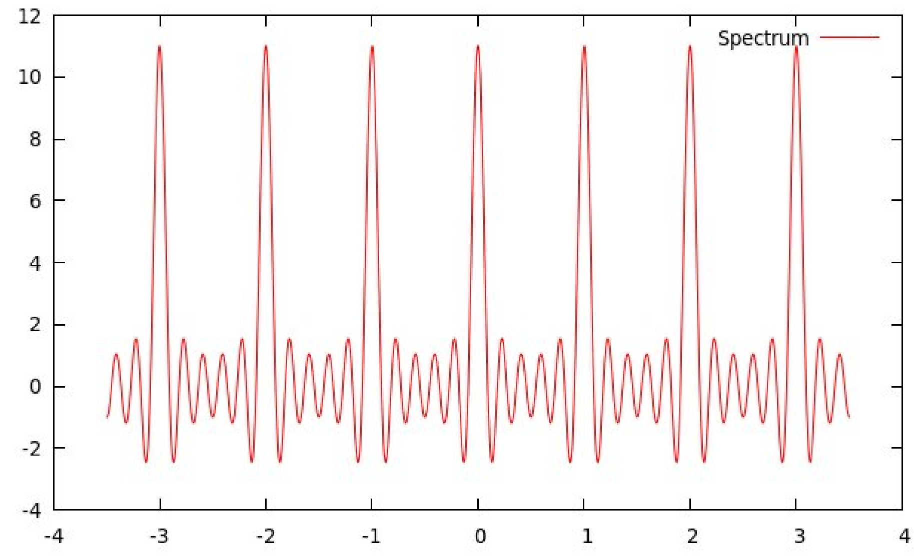
$$H(f) = \frac{sin 2\pi (N+1)/2 \frac{f}{f_0}}{sin \pi \frac{f}{f_0}}$$

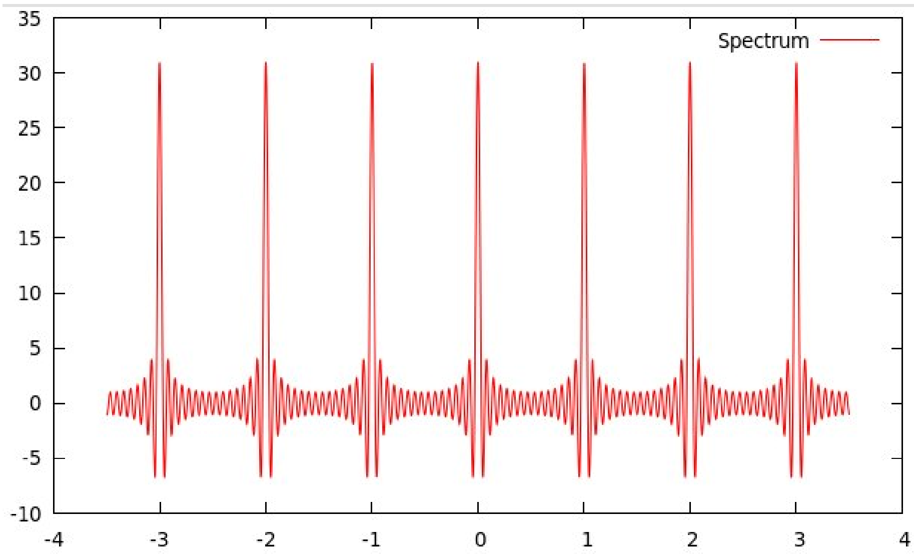
We could plot it in frequency domain as following

N = 3, 5



N = 10, 30


Spectrum
12
10
8
6
4
2
0
-2
-4
-4
-3
-2
-1
0
1
2
3
4


Spectrum
35
30
25
20
15
10
5
0
-5
-10
-4
-3
-2
-1
0
1
2
3
4

The source code is attached.

```c

/* Start reading here */
#include <fftw3.h>
#include "gnuplot_i.h"
#define NUM_POINTS 2096

/* Never mind this bit */
#include <stdio.h>
#include <math.h>

#define SLEEP_LGTH  1
#define TOL      0.0001

double h_linspan[NUM_POINTS];
double freq_linspan[NUM_POINTS];

void Hf(unsigned int N, double f[NUM_POINTS], double H[NUM_POINTS]) {
    int i;
    double den;
    for (i =0; i < NUM_POINTS; i++) {
        den = sin(M_PI*f[i]);
        H[i] = sin(M_PI*(N+1)*f[i]) / den;
    }
}

void freq_init(double freq_range[2], double out[NUM_POINTS]) {
    int i;
    double f = freq_range[0];
    double f_increment = (freq_range[1]-freq_range[0]) / NUM_POINTS;
    for (i =0; i < NUM_POINTS; i++, f += f_increment) {
        out[i] = f;
    }
}

/* Resume reading here */

int main(int argc, char *argv[]) {

    int N;
    if (argc >1) {
        N = atoi(argv[1]);
    } else {
        N = 3;
    }
    gnuplot_ctrl *h;

    /* Initialize the gnuplot handle */
    printf("*** DSP example of FFT gnuplot through C ***\n") ;
    h = gnuplot_init() ;


    /* Spectrum */
```

```
    double freq_range[2] = {-3.5, 3.5};

    freq_init(freq_range, freq_linspan);
    Hf(N, freq_linspan, h_linspan);

    /* Plot the signal */
    gnuplot_resetplot(h) ;
    gnuplot_setstyle(h, "lines") ;
    printf("\n\n*** Original Signal ***\n") ;
    gnuplot_plot_xy(h,
                    freq_linspan,
                    h_linspan,
                    NUM_POINTS,
                    "Spectrum") ;
    sleep(SLEEP_LGTH) ;

    printf("\n\n") ;
    printf("*** end of DSP example ***\n") ;
    /* Kill plotting Handler */
    gnuplot_resetplot(h);
    gnuplot_close(h);
    return 0;
}
```