



프로젝트 보고서

신호 및 시스템 컴퓨터 프로젝트 1

인하대학교 정보통신공학과

신호 및 시스템 (ICE3001)

2019-11-14

작성자: 김태인(12131671); 임세은(12181827)

목차

제시 조건	2
문제 (1): $t=0$ 에서 $t=2$ 에 대하여 $xN(t)$ 의 그래프를 그려라.	3
문제 (2): (1)의 4가지 경우에 대해 $eNt = xt - xN(t)$ 의 그래프를 그려라.	5
문제 (3): (1)의 4가지 경우에 대해 eNt 의 최대값을 구하고 Gibbs 현상에 대해 설명하라.	7
프로그램 코드	9

제시 조건

3.47. The signal of Figure P3.47 can be represented as

$$x(t) = \frac{4}{\pi} \sum_{\substack{n=1, \\ n=\text{odd}}}^{\infty} \frac{1}{n} \sin n\pi t$$

Using the approximation

$$\hat{x}_N(t) = \frac{4}{\pi} \sum_{\substack{n=1, \\ n=\text{odd}}}^N \frac{1}{n} \sin n\pi t$$

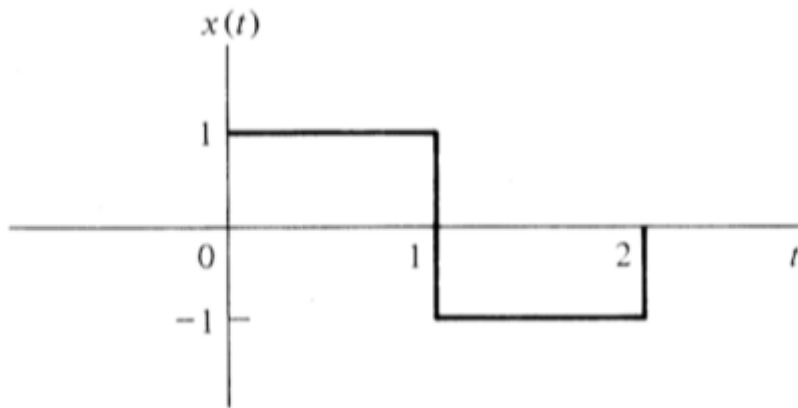


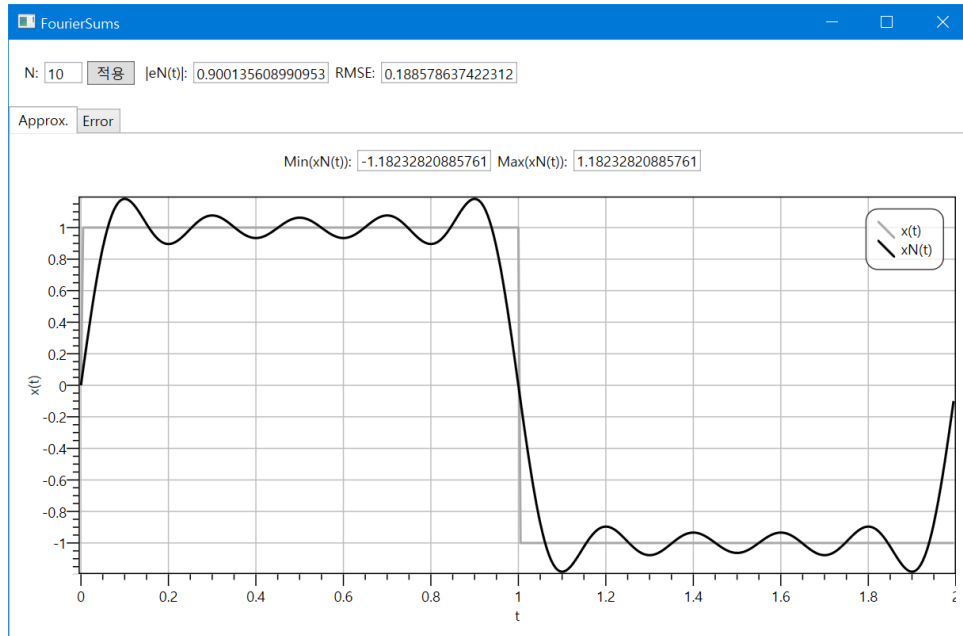
Figure P3.47

참고: 문제 (1)과 (2)에서 그래프를 그릴 때 $\Delta t = 0.005$ 를 사용하라.

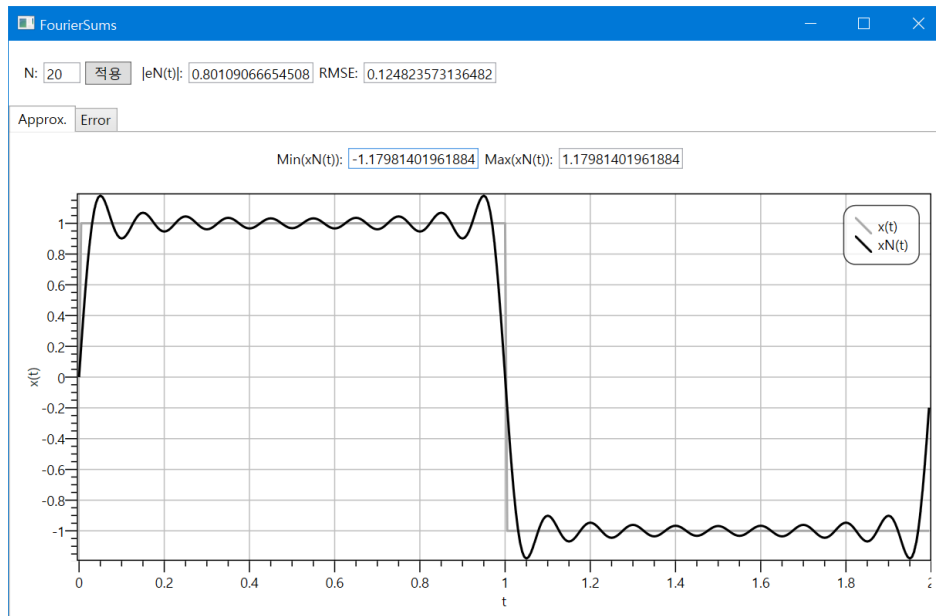
문제 (1): $t=0$ 에서 $t=2$ 에 대하여 $\hat{x}_N(t)$ 의 그래프를 그려라.

(단, N 의 값은 10, 20, 100, 500을 사용하라. 즉, 각각의 N 값에 대한 $\hat{x}_N(t)$ 의 그래프를 그려라.)

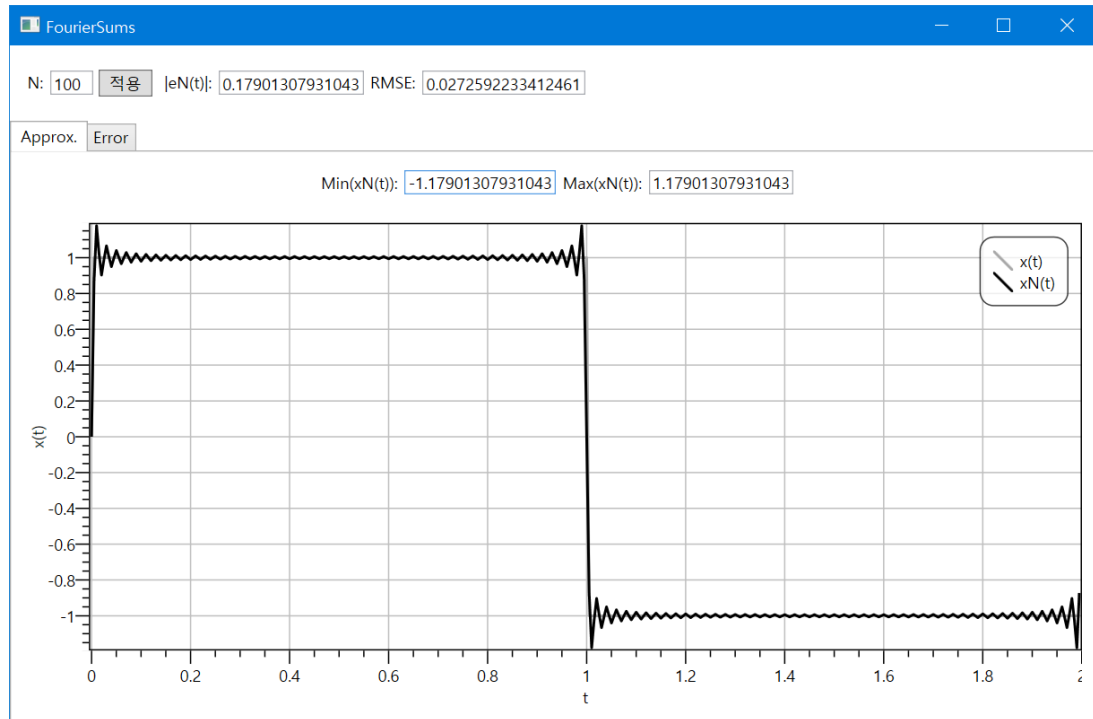
i. $N = 10$



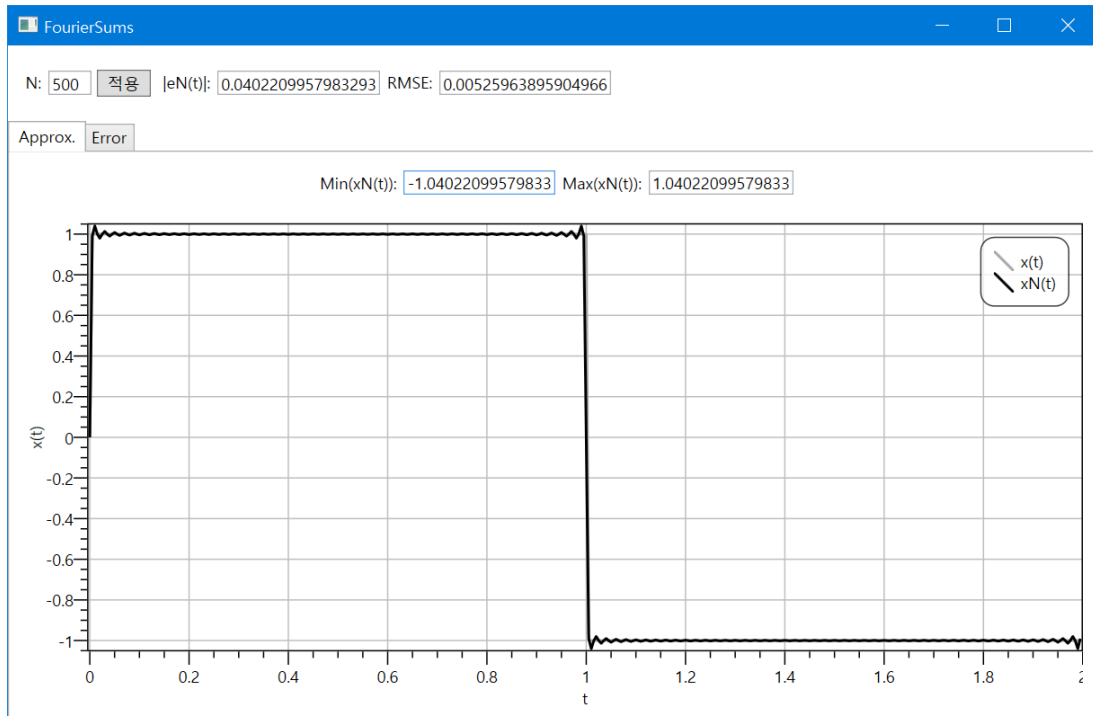
ii. $N = 20$



iii. $N = 100$

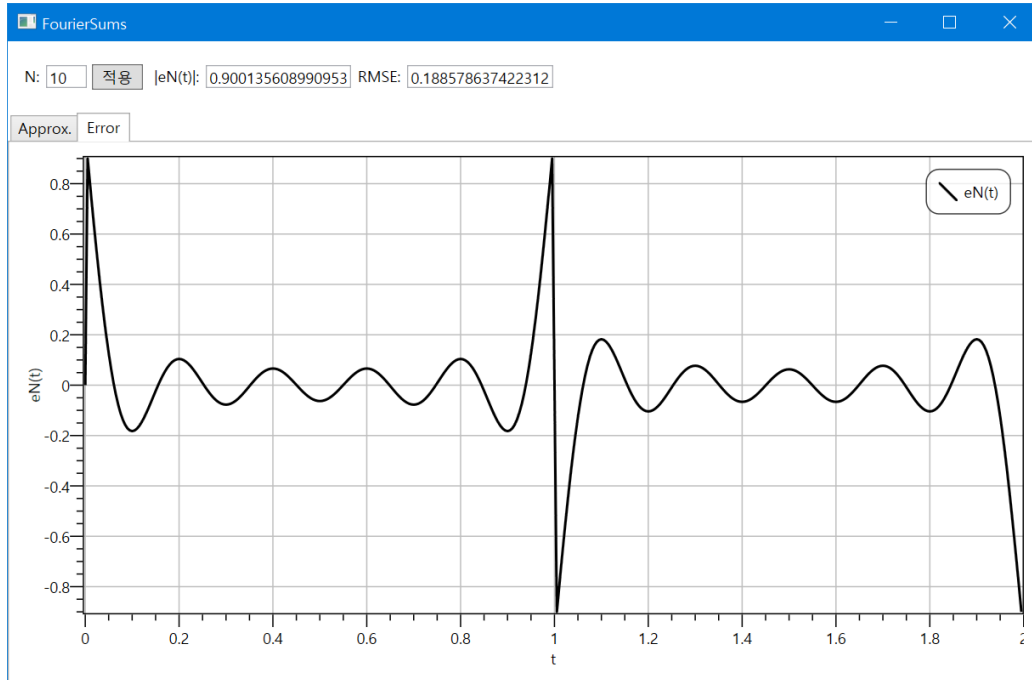


iv. $N = 500$

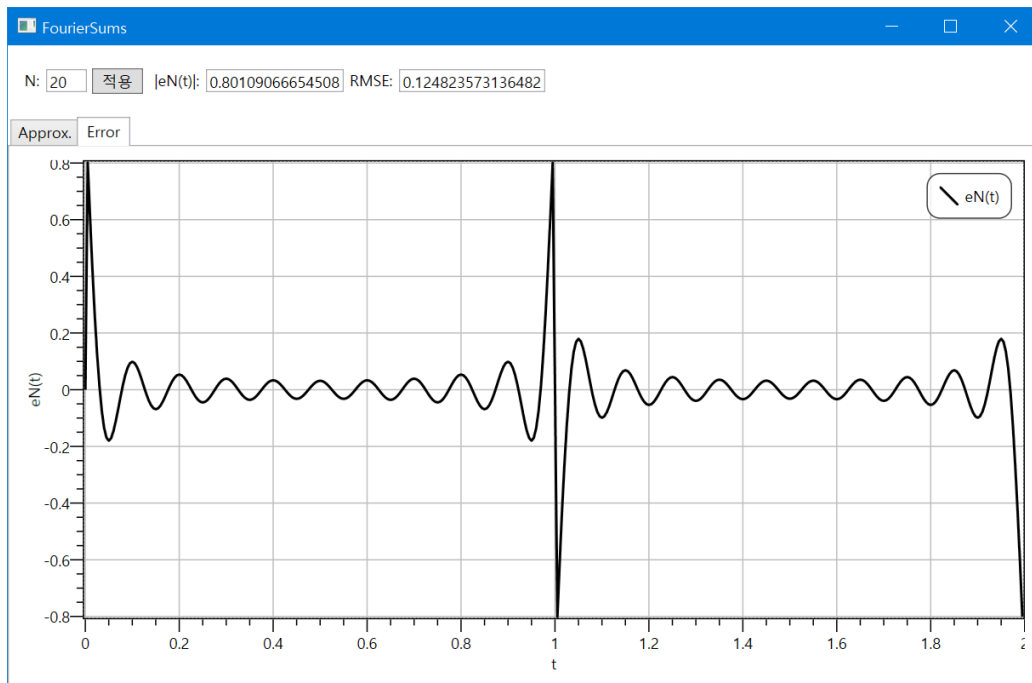


문제 (2): (1)의 4가지 경우에 대해 $e_N(t) = x(t) - \hat{x}_N(t)$ 의 그래프를 그려라.

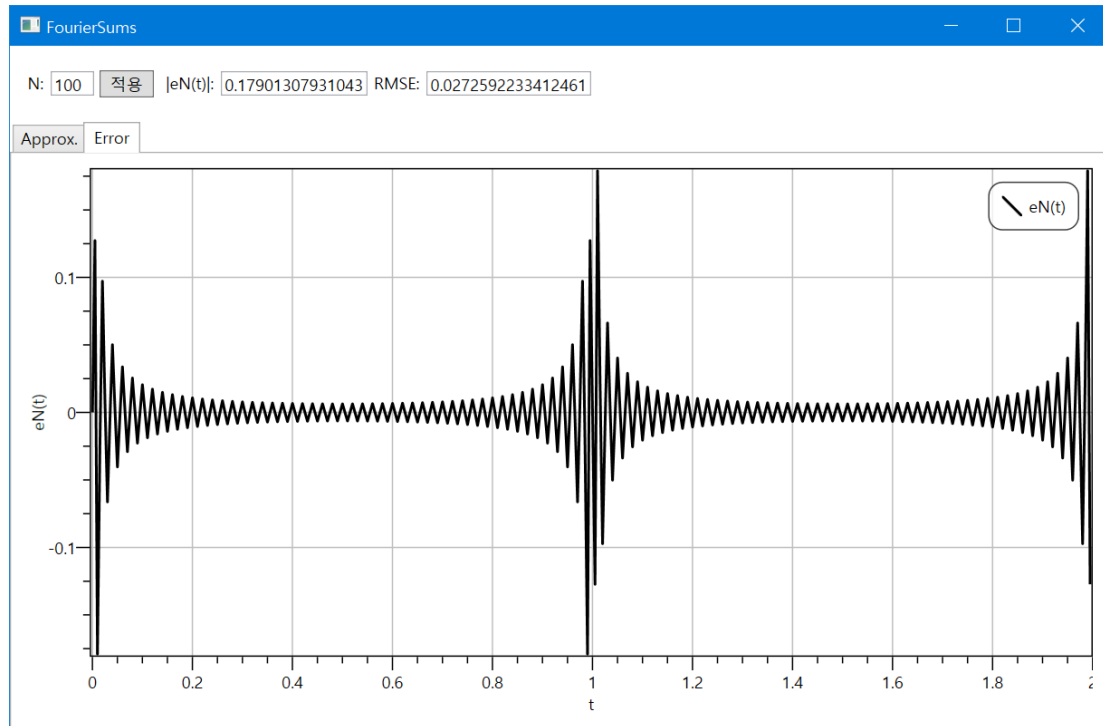
i. $N = 10$



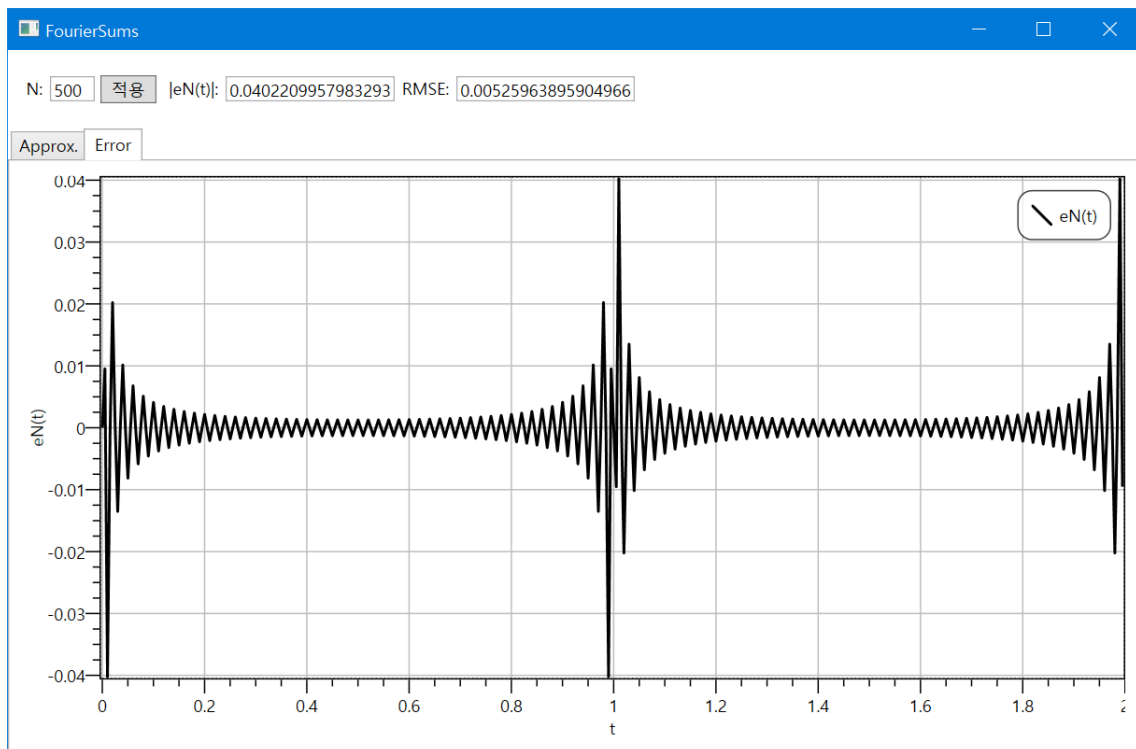
ii. $N = 20$



iii. $N = 100$



iv. $N = 500$



문제 (3):(1)의 4가지 경우에 대해 $|e_N(t)|$ 의 최대값을 구하고 Gibbs 현상에 대해 설명하라.

i. $N = 10$

N: 10 적용 $|e_N(t)|$: 0.900135608990953 RMSE: 0.188578637422312

ii. $N = 20$

N: 20 적용 $|e_N(t)|$: 0.80109066654508 RMSE: 0.124823573136482

iii. $N = 100$

N: 100 적용 $|e_N(t)|$: 0.17901307931043 RMSE: 0.0272592233412461

iv. $N = 500$

N: 500 적용 $|e_N(t)|$: 0.0402209957983293 RMSE: 0.00525963895904966

Gibbs 현상은 주기 함수를 푸리에 급수에 기반해 근사하는 과정에서 주기 함수의 불연속점 주변 점들을 근사한 값들이 항상 원래의 값보다 크게 나오는 현상을 말하며, 따라서 위에서 나열하는 $|e_N(t)|$ 이 0보다 큰 값에 수렴함을 알 수 있다.

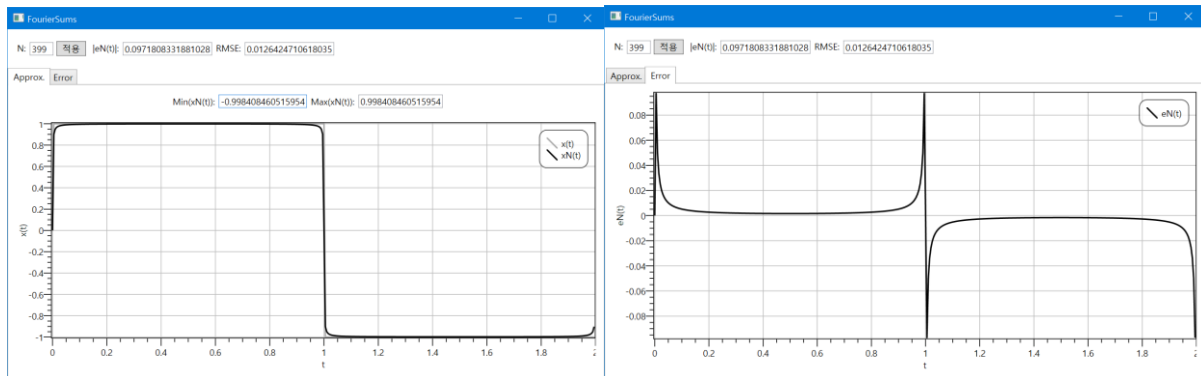
하지만 문제에서 제시한 N 값들로는 N 에 따른 $|e_N(t)|$ 의 변화 양상을 정확히 확인할 수 없어 임의로 N 을 변화시켜보았고, 다음 N 들을 통해 $[0, 144]$ 구간에서 $|e_N(t)|$ 이 최소값으로 약 0.09를 가짐을 알 수 있었다. 이는 Gibbs 현상이 예상하는 최소 오차에 근접한다.

N: 141	적용	$ e_N(t) $: 0.0892171357738643	RMSE: 0.0170662307129216
N: 143	적용	$ e_N(t) $: 0.0883494989457241	RMSE: 0.0169746725676136
N: 145	적용	$ e_N(t) $: 0.0950265951212721	RMSE: 0.0169177825936911
[0, 144]에서 $N=143$ 일 때 원본 신호와의 차이가 약 9%로 가장 적음			

하지만 N 을 144 이후 구간에서 N 을 증가시키면 $|e_N(t)|$ 이 다시 증가하다가 감소하는데, 이때 얻은 최소값은 약 0.06으로 N 이 143일 때 보다 더 적었다. 이는 컴퓨터에서 다루는 부동소수점을 합하는 과정에서 전파되는 오차로 인한 것이 아닐까 추측해본다.

N: 281	적용	$ e_N(t) $: 0.0627825095007182	RMSE: 0.0105421442030962
N: 283	적용	$ e_N(t) $: 0.0584428888351771	RMSE: 0.0103263167466948
N: 285	적용	$ e_N(t) $: 0.0595622974749401	RMSE: 0.010117729323817
[145, 284]에서 $N=283$ 일 때 원본 신호와의 차이가 약 6%로 가장 적음			

그리고 $N=399$ 일 때 아래와 같은 형태로 근사된 주기함수를 얻을 수 있었는데, 이전까지는 불연속점 부근에서 원래의 값보다 근사된 값이 컸지만 이 구간에선 근사된 값이 작은 것을 확인할 수 있었으며, $\hat{x}_N(t)$ 와 $e_N(t)$ 의 그래프가 모두 매끄러운 형태를 가지고 있었다. 하지만 이것이 어떤 이유로 나타난 것인지는 정확히 알지 못했으며, 역시 누적합의 오차가 전파되면서 나타난 현상이 아닐까 추측해본다.



(좌) $N=399$ 일 때 $\hat{x}_N(t)$ 의 그래프 (우) $N=399$ 일 때 $e_N(t)$ 의 그래프

참고로 본 과제에서는 불연속점에서의 $e_N(t)$ 는 집계하지 않았다. 항상 원본 함수는 불연속점에서 0을 갖지만 제시된 급수는 불연속점에서 +1 또는 -1을 가지게 되어 불연속점에서의 오차를 제외하지 않으면 문제에서 원하는 최대/최소 오차를 얻을 수 없었기 때문이다.

프로그램 코드

개발 환경

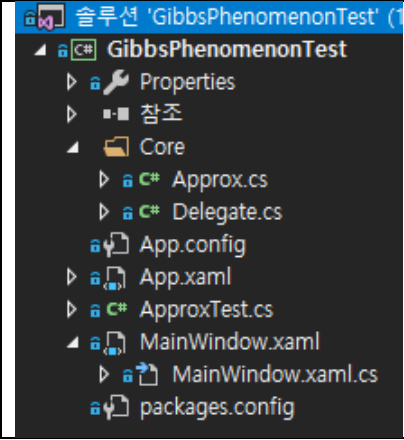
- 개발 프로그램: Visual Studio 2019
- 사용 언어: C#, XAML
- 프레임워크: .Net Framework, WPF
- 외부 라이브러리: InteractiveDataDisplay.WPF (그래프 작성 목적)

실행 환경

- 운영 체제: 64비트 기반 Microsoft Windows 7 이상
- 프레임워크: .NET Framework 4.7.2

코드 구조

(중요한 파일만 설명하였습니다)

	<pre>GibbsPhenomenonTest/ ├── /Core │ ├── Approx.cs: 수치해석에 필요한 메소드를 정의 │ ├── Delegate.cs: 인자로 전달할 함수의 형태를 정의 │ ├── ApproxTest.cs: 깁스현상을 확인하기 위한 함수 정의 │ └── MainWindow.xaml: 주 윈도우 창의 디자인을 정의 │ └── MainWindow.xaml.cs: 주 윈도우 UI의 행동 작성</pre>
---	---

Approx.cs

```
using GibbsPhenomenonTest.Core.Delegate;

namespace GibbsPhenomenonTest.Core
{
    public static class Approx
    {
        public static double[] GetDiscreteT(int from, int to, double delta_t)
        {
            {
                // 구간 [from, to]에 있는, delta_t만큼의 간격을 갖는 t들을 구함
                if (from > to) {
                    return new double[0];
                }
                uint count = (uint)((to - from) / delta_t);
                double[] result = new double[count];

                for (int i = 0; i < count; i++) {
                    result[i] = i * delta_t;
                }
                return result;
            }
        }

        public static double[] SetOf
            (Function1Continuous<double, double> x_t, int from, int to, double delta_t)
        {
            {
                // delta_t만큼의 간격을 갖는 구간 [from,to]에서의 연속함수 x(t)값을 모두 구함
                if (from > to) { return new double[0]; }

                var dis_t = GetDiscreteT(from, to, delta_t);
                double[] f_t = new double[dis_t.Length];

                for (int i = 0; i < dis_t.Length; i++) {
                    f_t[i] = x_t(dis_t[i]);
                }

                return f_t;
            }
        }

        public static double[] SetOf
            (Function1Discrete<double, double, uint> xN_t, int from, int to, double delta_t, uint N)
        {
            {
                // delta_t만큼의 간격을 갖는 구간 [from,to]에서의 이산함수 xN(t)값을 모두 구함
                if (from > to) { return new double[0]; }

                var dis_t = GetDiscreteT(from, to, delta_t);
                double[] f_t = new double[dis_t.Length];

                for (int i = 0; i < dis_t.Length; i++) {
                    f_t[i] = xN_t(dis_t[i], N);
                }

                return f_t;
            }
        }
    }
}
```

Delegate.cs

```
namespace GibbsPhenomenonTest.Core
{
    namespace Delegate
    {
        { // 함수를 다른 함수의 인자로 넘기기 위해 정의한 delegate
            // 연속함수 f(x)
            public delegate B Function1Continuous<B, A1>(A1 A);

            // 이산함수 f(n) , N=SIGMA N의 최대값
            public delegate B Function1Discrete<B, A1, N1>(A1 A, N1 N);
        }
    }
}
```

ApproxTest.cs

```
using System;
using GibbsPhenomenonTest.Core.Delegate;

namespace GibbsPhenomenonTest
{
    public class ApproxTest
    {
        // Gibbs 현상을 확인하기 위한 속성과 행동이 정의된 클래스
        public ApproxTest(uint N, double delta_t)
        {
            if (N != 0) {
                this.N = N;
            }
            else {
                this.N = 1;
            }

            if (delta_t > 0) {
                Delta_t = delta_t;
            }
            else {
                Delta_t = 0.001;
            }
        }

        public uint N {
            get; set;
        }

        public double Delta_t {
            get; set;
        }

        public Function1Continuous<double, double> x_t = t =>
        {
            // Square Wave의 연속함수 형태 (과제에서 제시된 무한급수 형태는 이용할 수 없기 때문)
            return Math.Sign(Math.Sin(Math.PI * t));
        };

        public Function1Discrete<double, double, uint> xN_t =
        (t, NMax) =>
        {
            // Square Wave의 이산함수 형태
            double result = 0;
            for (int n = 1; n <= NMax; n += 2)
            {
                result += (1 / (double)n) * Math.Sin(n * Math.PI * t);
            }
            result *= (4 / Math.PI);

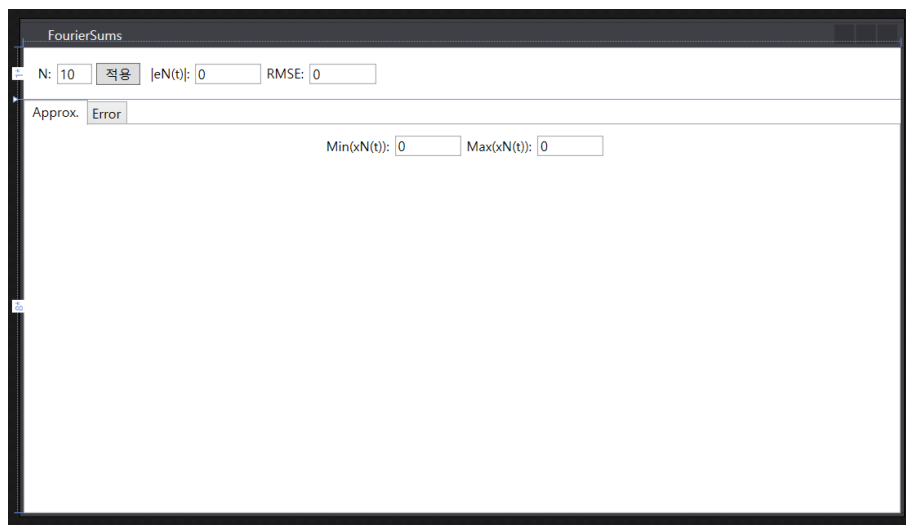
            return result;
        };
    }
}
```

MainWindow.xaml

```
<Window x:Class="GibbsPhenomenonTest.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:d3="clr-namespace:InteractiveDataDisplay.WPF;assembly=InteractiveDataDisplay.WPF"
        mc:Ignorable="d"
        Title="FourierSums" Height="450" Width="800">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="1*"/>
            <RowDefinition Height="8*"/>
        </Grid.RowDefinitions>
        <Grid Grid.Row="0" Margin="8">
            <StackPanel Orientation="Horizontal"
                        HorizontalAlignment="Left"
                        VerticalAlignment="Center">
                <Label Content="N:"/>
                <TextBox Name="tbN"
                        Text="10" MinWidth="32"
                        VerticalAlignment="Center"
                        KeyDown="TbN_KeyDown"/>
                <Button Name="btApply"
                        Content="적용"
                        Margin="4"
                        MinWidth="40"
                        Click="BtApply_Click"/>
                <Label Content="|eN(t)|:"/>
                <TextBox Name="tbAbsError"
                        Text="0" MinWidth="60"
                        IsReadOnly="True"
                        VerticalAlignment="Center"/>
                <Label Content="RMSE:"/>
                <TextBox Name="tbRMSEError"
                        Text="0" MinWidth="60"
                        IsReadOnly="True"
                        VerticalAlignment="Center"/>
            </StackPanel>
        </Grid>
        <Grid Grid.Row="1">
```

```

            <TabControl>
                <TabItem Header="Approx.">
                    <Grid>
                        <Grid.RowDefinitions>
                            <RowDefinition Height="1*"/>
                            <RowDefinition Height="9*"/>
                        </Grid.RowDefinitions>
                        <StackPanel Grid.Row="0"
                                    Orientation="Horizontal"
                                    VerticalAlignment="Center"
                                    HorizontalAlignment="Center">
                            <Label Content="Min(xN(t)):"/>
                            <TextBox Name="tbMin"
                                    Text="0" MinWidth="60"
                                    IsReadOnly="True"
                                    VerticalAlignment="Center"/>
                            <Label Content="Max(xN(t)):"/>
                            <TextBox Name="tbMax"
                                    Text="0" MinWidth="60"
                                    IsReadOnly="True"
                                    VerticalAlignment="Center"/>
                        </StackPanel>
                        <d3:Chart Grid.Row="1" BottomTitle="t"
                                LeftTitle="x(t)"
                                Margin="8">
                            <Grid Name="Graphs"/>
                        </d3:Chart>
                    </Grid>
                </TabItem>
                <TabItem Header="Error">
                    <d3:Chart BottomTitle="t" LeftTitle="eN(t)"
                                Margin="8">
                        <d3:LineGraph Name="eNGraph"
                                    Description="eN(t)"
                                    Stroke="Black"
                                    StrokeThickness="2"/>
                    </d3:Chart>
                </TabItem>
            </TabControl>
        </Grid>
    </Window>
```



MainWindow.xaml.cs

```
using GibbsPhenomenonTest.Core;
using InteractiveDataDisplay.WPF;
using System;
using System.Linq;
using System.Windows;
using System.Windows.Input;
using System.Windows.Media;

namespace GibbsPhenomenonTest
{
    /// <summary>
    /// MainWindow.xaml에 대한 상호 작용 논리
    /// </summary>
    public partial class MainWindow : Window
    {
        private readonly ApproxTest test;
        private const int FROM = 0;
        private const int TO = 2;
        private double[] dis_t;
        private double[] x_t;
        private double[] xN_t;
        private double[] eN_t;

        private LineGraph graph_x_t;
        private LineGraph graph_xN_t;

        public MainWindow()
        {
            test = new ApproxTest(10, 0.005);
            dis_t = Approx.GetDiscreteT(FROM, TO, test.Delta_t);

            InitializeComponent();
            tbN.Text = test.N.ToString();
            InitializeApproxTab();
            InitializeErrorTab();

            UpdateMinMax_xN_t();
            UpdateErrors();
        }

        private void UpdateMinMax_xN_t()
        {
            tbMin.Text = xN_t.Min().ToString();
            tbMax.Text = xN_t.Max().ToString();
        }

        private void UpdateErrors()
        {
            UpdateAbsErr();
            UpdateRMSErr();
        }
    }
}
```

```

private void UpdateRMSErr()
{
    double RMSE =
        Math.Sqrt(
            eN_t.Sum((e) => Math.Pow(e, 2))
            / eN_t.Length
        );
    tbRMSError.Text = RMSE.ToString();
}

private void UpdateAbsErr()
{
    /*
     * eN_t : -1, +0.4, -0.2, +2
     * eN_t.Min() == -1 => +1
     * eN_t.Max() == +2 => +2
     * 각각의 절댓값을 구하면?
     */
    var absMin = Math.Abs(eN_t.Min());
    var absMax = Math.Abs(eN_t.Max());
    tbAbsError.Text = (absMin > absMax ? absMin : absMax).ToString();
}

private void InitializeErrorTab()
{
    UpdateErrorGraphs();
}

private void InitializeApproxTab()
{
    x_t = Approx.SetOf(test.x_t, FROM, T0, test.Delta_t);
    xN_t = Approx.SetOf(test.xN_t, FROM, T0, test.Delta_t, test.N);

    graph_x_t = new LineGraph();
    Graphs.Children.Add(graph_x_t);
    graph_x_t.Description = "x(t)";
    graph_x_t.Stroke = Brushes.DarkGray;
    graph_x_t.StrokeThickness = 2;
    graph_x_t.Plot(dis_t, x_t);

    graph_xN_t = new LineGraph();
    Graphs.Children.Add(graph_xN_t);
    graph_xN_t.Description = "xN(t)";
    graph_xN_t.Stroke = Brushes.Black;
    graph_xN_t.StrokeThickness = 2;
    graph_xN_t.Plot(dis_t, xN_t);
}

private void BtApply_Click(object sender, RoutedEventArgs e)
{
    ApplyChanges();
}

```



```

private void ApplyChanges()
{
    if (uint.TryParse(tbN.Text, out uint result) && result != 0) {
        test.N = result;
        UpdateGraphs();
        UpdateErrors();
        UpdateMinMax_xN_t();
    }
    else {
        tbN.Text = test.N.ToString();
    }
}

private void UpdateGraphs()
{
    UpdateApproxGraphs();
    UpdateErrorGraphs();
}

private void UpdateErrorGraphs()
{
    eN_t = x_t.Zip(xN_t, (a, b) => a - b).ToArray();

    // distance: t 축에서 1만큼의 거리가 delta_t 간격으로 얼마나 나뉘었는지
    uint distance = (uint)(1.0 / test.Delta_t);
    for (uint i = 0; i < eN_t.Length; i += distance)
    {
        // 시작점과 끝점이 정수이므로 불연속점의 위치를 알 수 있음
        eN_t[i] = 0; // 불연속점에서의 오차는 생략 처리함
    }

    eNGraph.Plot(dis_t, eN_t);
}

private void UpdateApproxGraphs()
{
    dis_t = Approx.GetDiscreteT(FROM, TO, test.Delta_t);
    x_t = Approx.SetOf(test.x_t, FROM, TO, test.Delta_t);
    xN_t = Approx.SetOf(test.xN_t, FROM, TO, test.Delta_t, test.N);
    graph_x_t.Plot(dis_t, x_t);
    graph_xN_t.Plot(dis_t, xN_t);
}

private void TbN_KeyDown(object sender, KeyEventArgs e)
{
    if (e.Key == Key.Enter) {
        ApplyChanges();
    }
}
}

```