実技課題　　解答及び解説

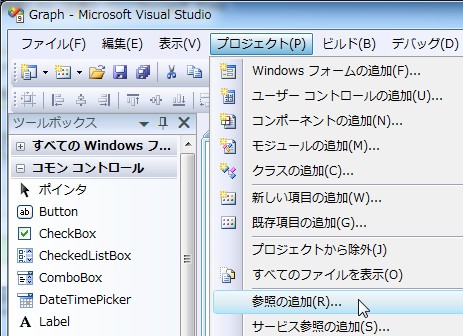
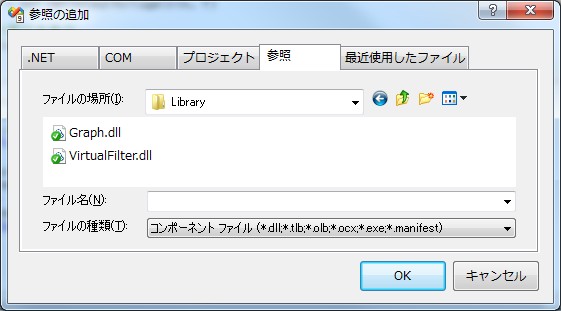
実技課題　解答及び解説

「パソコンを用いた計測制御システムの製作（Visual C#）」

　解答例を以下に示す。

（１）事前の設定(別添のファイルに格納)

1. プロジェクトを開いたあと、グラフ描画用のDLL(Graph.dll)をメニューバーの「プロジェクト」→「参照の追加」から以下のダイアログを表示し、「参照」のタブで、配布した"Graph.dll"を選択する。

1. "Polytech.Windows.Form"名前空間でGraphクラスを定義しているため、Usingステートメントで"Polytech.Windows.Form"名前空間を追加すること。

using System;

using System.Drawing;

using System.Drawing.Drawing2D;

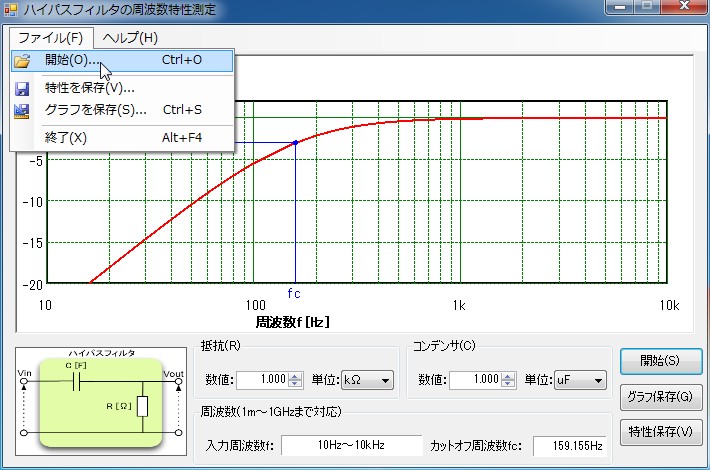
using System.Drawing.Imaging;

using Polytech.Windows.Form; //Graphクラス

using Filter; //HighPassFilterクラス

1. コントロールと各機能

|  |  |  |
| --- | --- | --- |
| コントロール | オブジェクト名 | 備考 |
| Form | Form1 | Graphクラスの初期化と軸の描画、初期描画の登録 |
| Button | GraphInit | 初期描画状態を呼び出す |
| SaveCSV | 周波数特性をCSVファイル等で保存 |
| SaveGraph | グラフを画像ファイルとして保存 |
| NumericUpdown | InputNumberR | 抵抗値の数値を入力 |
| InputNumberC | 静電容量の数値を入力 |
| ListBox | ListPrefixR | 抵抗値の単位を入力 |
| ListPrefixC | 静電容量の単位を入力 |
| Label | SimlateShowFreq | カットオフ周波数の理論値を表示 |
| MeasureShowFreq | カットオフ周波数の測定値を表示 |
| DifferenceFreqPercemt | 理論値と測定値の誤差を%表示 |
| PictureBox | GraphDisplay | グラフの描画対象 |



解答例のソースコード(フォーム)

|  |  |
| --- | --- |
| 行 | プログラム |
|  | using System; |
|  | using System.IO; |
|  | using System.Collections.Generic; |
|  | using System.ComponentModel; |
|  | using System.Data; |
|  | using System.Drawing; |
|  | using System.Linq; |
|  | using System.Text; |
|  | using System.Drawing.Drawing2D; |
|  | using System.Drawing.Imaging; |
|  | using System.Windows.Forms; |
|  | using Polytech.Windows.Form; //Graphクラス |
|  | using Filter; //HighPassFilterクラス |
|  |  |
|  | namespace GraphFilterSample |
|  | { |
|  | public partial class GraphForm : Form |
|  | { |
|  | //フォーム内で使用するメンバー |
|  |  |
|  | static private Graph myGraph; //グラフ描画用オブジェクト |
|  | static private decimal PartR, PartC; //抵抗値, 静電容量 |
|  |  |
|  | static private PointF CutoffPoint; //カットオフ周波数の座標 |
|  |  |
|  | static private readonly float GvRangeMin = -20; //Y軸最小値 |
|  | static private readonly float GvRangeMax = 2; //Y軸最大値 |
|  | static private readonly float FreqRangeMin = (float)Math.Pow(10, -3);  //Y軸最小値 |
|  | static private readonly float FreqRangeMax = (float)Math.Pow(10, 9);  //Y軸最大値 |
|  | static private readonly float Vin = 1; //入力電圧 |
|  |  |
|  | static private List<PointF> Series = new List<PointF>(); //グラフの系列 |
|  | static private HighPassFilter hpf; //測定対象のハイパスフィルタ |
|  |  |
|  | public GraphForm() |
|  | { |
|  | InitializeComponent(); |
|  | //イベントハンドラの追加 |
|  | Load += new EventHandler(GraphForm\_Load); |
|  | FormClosing += new FormClosingEventHandler(GraphForm\_FormClosing); |
|  |  |
|  | DrawGraph.Click += new EventHandler(DrawGraph\_Click); |
|  | MenuFileOpen.Click += new EventHandler(DrawGraph\_Click); |
|  |  |
|  | SaveGraph.Click +=new EventHandler(SaveGraph\_Click); |
|  | MenuFileSaveGraph.Click += new EventHandler(SaveGraph\_Click); |
|  |  |
|  | SaveCSV.Click += new EventHandler(SaveCSV\_Click); |
|  | MenuFileSaveCSV.Click += new EventHandler(SaveCSV\_Click); |
|  |  |
|  | MenuCircuitView.Click += new EventHandler(MenuCircuitView\_Click); |
|  |  |
|  | } |
|  |  |
|  | private void GraphForm\_Load(System.Object sender, System.EventArgs e) |
|  | { |
|  | //ピクチャーボックスの外形線を立体表示 |
|  | GraphDisplay.BorderStyle = BorderStyle.Fixed3D; |
|  |  |
|  | //抵抗の単位のリスト追加 |
|  | ListPrefixR.Items.Add("Ω"); |
|  | ListPrefixR.Items.Add("kΩ"); |
|  | ListPrefixR.Items.Add("MΩ"); |
|  | ListPrefixR.SelectedIndex = 1; |
|  | //抵抗値の入力範囲の設定 |
|  | InputNumberR.Value = 1; |
|  | InputNumberR.Maximum = (decimal)Math.Pow(10, 5); |
|  | InputNumberR.Minimum = (decimal)Math.Pow(10, -3); |
|  | InputNumberR.DecimalPlaces = 3; |
|  | InputNumberR.Increment = 0.01M; |
|  |  |
|  | //静電容量の単位のリスト追加 |
|  | ListPrefixC.Items.Add("pF"); |
|  | ListPrefixC.Items.Add("nF"); |
|  | ListPrefixC.Items.Add("uF"); |
|  | ListPrefixC.SelectedIndex = 2; |
|  | //静電容量の入力範囲の設定 |
|  | InputNumberC.Value = 1; |
|  | InputNumberC.Maximum = (decimal)Math.Pow(10, 5); |
|  | InputNumberC.Minimum = (decimal)Math.Pow(10, -3); |
|  | InputNumberC.DecimalPlaces = 3; |
|  | InputNumberC.Increment = 0.01M; |
|  |  |
|  | //イベントの関連付け(上記の設定変更でイベントが動作しないようにするため) |
|  | InputNumberR.ValueChanged += new EventHandler(InputParameter); |
|  | InputNumberC.ValueChanged += new EventHandler(InputParameter); |
|  | ListPrefixR.SelectedValueChanged += new EventHandler(InputParameter); |
|  | ListPrefixC.SelectedValueChanged += new EventHandler(InputParameter); |
|  |  |
|  | //パラメータ入力のイベントを呼び出す |
|  | InputParameter(sender, e); |
|  |  |
|  | } |
|  |  |
|  | /// <summary>起動時に実行されるイベントです。</summary> |
|  | private void GraphForm\_FormClosing(System.Object sender,  System.Windows.Forms.FormClosingEventArgs e) |
|  | { |
|  | DialogResult result; |
|  |  |
|  | result = MessageBox.Show("プログラムを終了しますか?", "確認", MessageBoxButtons.YesNo, MessageBoxIcon.Question); |
|  | if (result == DialogResult.No) |
|  | { |
|  | e.Cancel = true; //このイベントをキャンセル |
|  | } |
|  | } |
|  |  |
|  | /// <summary>抵抗RとコンデンサCの入力された数値と単位からパラメータを  演算するメソッドです。</summary> |
|  | private void InputParameter(System.Object sender, System.EventArgs e) |
|  | { |
|  | //入力されたRとCが0以下の時は以下を実行しない |
|  | if ((InputNumberR.Value <= 0) || (InputNumberC.Value <= 0)) |
|  | return; |
|  |  |
|  | switch (ListPrefixR.Text) |
|  | { |
|  | case "MΩ": |
|  | PartR = InputNumberR.Value \* (decimal)Math.Pow(10, 6); |
|  | break; |
|  | case "kΩ": |
|  | PartR = InputNumberR.Value \* (decimal)Math.Pow(10, 3); |
|  | break; |
|  | default: |
|  | PartR = InputNumberR.Value; |
|  | break; |
|  | } |
|  |  |
|  | switch (ListPrefixC.Text) |
|  | { |
|  | case "pF": |
|  | PartC = InputNumberC.Value \* (decimal)Math.Pow(10, -12); |
|  | break; |
|  | case "nF": |
|  | PartC = InputNumberC.Value \* (decimal)Math.Pow(10, -9); |
|  | break; |
|  | case "uF": |
|  | PartC = InputNumberC.Value \* (decimal)Math.Pow(10, -6); |
|  | break; |
|  | default: |
|  | PartC = InputNumberC.Value; |
|  | break; |
|  | } |
|  |  |
|  | } |
|  | /// <summary>周波数特性の実測値の描画とカットオフ周波数を開始する  イベントです。</summary> |
|  | private void DrawGraph\_Click(Object sender, System.EventArgs e) |
|  | { |
|  | int i = 0; |
|  | float freqMin = 1; //最小周波数 |
|  | float freqMax = 1; //最大周波数 |
|  | List<float> freqList = new List<float>(); //ファイルから読み込んだ周波数 |
|  | //グラフに適用する周波数の最小と最大値 |
|  | float graphFreqMin, graphFreqMax; |
|  |  |
|  | //①ファイル読み込み最小、最大周波数と周波数のListの取得 |
|  | if (!OpenCSV(ref freqMin, ref freqMax, ref freqList)) return; |
|  |  |
|  | //②測定対象のハイパスフィルタのインスタンス生成 |
|  | hpf = new HighPassFilter(PartR, PartC); |
|  | //抵抗とコンデンサのパラメータ |
|  | CutoffPoint.X = (float)hpf.GetCutoffFrequency(); |
|  | //カットオフ周波数の取得 |
|  | CutoffPoint.Y = (float)hpf.GetGain((decimal)Vin,  (decimal)CutoffPoint.X); |
|  | //利得の計算 |
|  | CutoffFreqLabel.Text = AddPrefixUnit(CutoffPoint.X, "Hz"); |
|  |  |
|  | //③周波数の範囲を表示 |
|  | FreqRangeLabel.Text = AddPrefixUnit(freqMin, "Hz") + "～" +  AddPrefixUnit(freqMax, "Hz"); |
|  | //周波数の範囲を10の乗数になるように変換してからグラフを初期化 |
|  | graphFreqMin = (float)(Math.Pow(10, Math.Floor(Math.Log10(freqMin)))); |
|  | graphFreqMax = (float)(Math.Pow(10,  Math.Ceiling(Math.Log10(freqMax)))); |
|  | DrawSheet(graphFreqMin, graphFreqMax); |
|  |  |
|  | //④描画データの生成 |
|  | Series.Clear(); |
|  | //グラフの座標のリストをクリア |
|  | //電圧利得を算出して系列データに代入 |
|  | foreach (float freq in freqList) |
|  | { |
|  | Series.Add(new PointF(freq, (float)hpf.GetGain((decimal)Vin,  (decimal)freq))); |
|  | } |
|  |  |
|  | //⑤系列データ間を線分で描画 |
|  | myGraph.LineSet(DashStyle.Solid, 2, Color.Red); |
|  | for (i = 0; i <= Series.Count - 2; i++) |
|  | { |
|  | if (GvRangeMin <= Series[i].Y & Series[i].Y <= GvRangeMax) |
|  | { |
|  | myGraph.DrawLine(Series[i], Series[i + 1]); |
|  | } |
|  | } |
|  |  |
|  | //⑥カットオフ周波数の描画 |
|  | myGraph.MarkerSet(Graph.MarkerShapeStyle.Circle, 5, Color.Blue); |
|  | myGraph.DrawMarker(CutoffPoint); |
|  | myGraph.LineSet(DashStyle.Solid, 1, Color.Blue); |
|  | myGraph.DrawLine(CutoffPoint.X, GvRangeMin, CutoffPoint.X,  CutoffPoint.Y); |
|  | myGraph.DrawLine(graphFreqMin, CutoffPoint.Y, CutoffPoint.X,  CutoffPoint.Y); |
|  | myGraph.FontColor = Color.Blue; |
|  | myGraph.DrawText(CutoffPoint.X \* 0.9F, GvRangeMin - 0.5F, "fc"); |
|  | myGraph.DrawText(graphFreqMin \* 0.7F, -3F + 0.5F,  AddPrefixUnit(-3).PadLeft(4)); |
|  | } |
|  |  |
|  | /// <summary>グラフを画像ファイルとして保存するメソッドです。</summary> |
|  | private void SaveGraph\_Click(Object sender, System.EventArgs e) |
|  | { |
|  | SaveFileDialog saveDialog = new SaveFileDialog(); |
|  | //ダイアログのタイトル |
|  | saveDialog.Title = "グラフの保存先を指定してください。"; |
|  | //デフォルトのファイル名は"年月日\_時分秒.bmp" |
|  | saveDialog.FileName = DateTime.Now.ToString("yyyyMMdd\_HHmmss") +  ".bmp"; |
|  | //デスクトップをデフォルトの保存先に指定 |
|  | saveDialog.InitialDirectory =  System.Environment.SpecialFolder.Desktop.ToString(); |
|  | //ファイルの種類はBMP, JPEG, PNG |
|  | saveDialog.Filter = "BMPファイル(\*.bmp)|\*.bmp|JPEGファイル  (\*.jpg;\*jpeg)|\*.jpg;\*jpeg|PNGファイル(\*.png)|\*.png"; |
|  | //ダイアログの表示がOK以外の時は以下の処理を実行しない |
|  | if (saveDialog.ShowDialog() != System.Windows.Forms.DialogResult.OK) |
|  | return; |
|  |  |
|  | if (!myGraph.Export(saveDialog.FileName)) |
|  | { |
|  | MessageBox.Show("保存するファイル形式、保存先が正しくありません。"); |
|  | } |
|  | } |
|  |  |
|  | /// <summary>ダイアログを表示して周波数のファイルを読み込みます。</summary> |
|  | /// <param name="fMin">最小周波数を参照渡しで取得します。</param> |
|  | /// <param name="fMax">最大周波数を参照渡しで取得します。</param> |
|  | /// <param name="fList">周波数をListクラスで取得します。</param> |
|  | /// <returns>ファイルを正常に読み込めた場合はtrueを、  読み込めない場合はfalseを返します。</returns> |
|  | private bool OpenCSV(ref float fMin, ref float fMax, ref List<float> fList) |
|  | { |
|  | //ファイルから読み込むためのダイアログ |
|  | OpenFileDialog openDialog = new OpenFileDialog(); |
|  |  |
|  | fList = new List<float>(); //ファイルから読み込んだ周波数 |
|  |  |
|  | //ダイアログのタイトル |
|  | openDialog.Title = "入力信号用のファイルを読み込んでください。"; |
|  | //デフォルトのファイル名指定なし |
|  | openDialog.FileName = ""; |
|  | //デスクトップをデフォルトのフォルダに指定 |
|  | openDialog.InitialDirectory =  System.Environment.SpecialFolder.Desktop.ToString(); |
|  | //ファイルの種類はCSVとText,すべてのファイル |
|  | openDialog.Filter = "CSVファイル|\*.csv|テキストファイル|\*.txt|  すべてのファイル|\*.\*"; |
|  | //ダイアログの表示がOK以外の時は以下の処理を実行しない |
|  | if (openDialog.ShowDialog() != System.Windows.Forms.DialogResult.OK) [return false; |
|  |  |
|  | //ファイルを読み込む処理 |
|  | using (StreamReader sr = new StreamReader(openDialog.FileName)) |
|  | { |
|  | //ファイルを末尾まで読み込む |
|  | while (!(sr.EndOfStream)) |
|  | { |
|  | float f = 0; //ファイルから読み込む数値 |
|  | //変換可能な文字列なら変換して格納 |
|  | if (float.TryParse(sr.ReadLine(), out f)) |
|  | { |
|  | fList.Add(f); //周波数の値を追加 |
|  | } |
|  | } |
|  | } |
|  |  |
|  | if (fList.Count < 2) |
|  | { |
|  | MessageBox.Show("読み込んだファイルのデータに不足、または異常が  あります。", "警告", MessageBoxButtons.OK, MessageBoxIcon.Exclamation); |
|  | return false; |
|  | } |
|  |  |
|  | fList.Sort(); //周波数のリストを昇順でソート |
|  |  |
|  | //周波数の最小値と最大値を格納 |
|  | fMin = fList[0]; |
|  | fMax = fList[fList.Count - 1]; |
|  |  |
|  | //周波数の測定範囲をチェック |
|  | if (fMin < FreqRangeMin || FreqRangeMax < fMax) |
|  | { |
|  | MessageBox.Show("周波数が描画可能が範囲を超えています。", "警告",  MessageBoxButtons.OK, MessageBoxIcon.Exclamation); |
|  | return false; |
|  | } |
|  | return true; |
|  | } |
|  |  |
|  | /// <summary>パラメータと系列データをCSVファイルに保存する  イベントです。</summary> |
|  | private void SaveCSV\_Click(Object sender, System.EventArgs e) |
|  | { |
|  | SaveFileDialog saveDialog = new SaveFileDialog(); |
|  | //ダイアログのタイトル |
|  | saveDialog.Title = "周波数特性の保存先を指定してください。"; |
|  | //デフォルトのファイル名は"年月日\_時分秒.csv" |
|  | saveDialog.FileName = DateTime.Now.ToString("yyyyMMdd\_HHmmss") +  ".csv"; |
|  | //デスクトップをデフォルトの保存先に指定 |
|  | saveDialog.InitialDirectory =  System.Environment.SpecialFolder.Desktop.ToString(); |
|  | //ファイルの種類はCSVとText,すべてのファイル |
|  | saveDialog.Filter = "CSVファイル|\*.csv|テキストファイル|\*.txt|  すべてのファイル|\*.\*"; |
|  | //ダイアログの表示がOK以外の時は以下の処理を実行しない |
|  | if (saveDialog.ShowDialog() != System.Windows.Forms.DialogResult.OK)  return; |
|  |  |
|  | using (StreamWriter sw = new StreamWriter(saveDialog.FileName)) |
|  | { |
|  | //ファイルの先頭にパラメータを記述 |
|  | sw.WriteLine("R=" + PartR.ToString()); |
|  | sw.WriteLine("C=" + PartC.ToString()); |
|  | //特性をファイルへ書き込む |
|  | foreach (PointF p in Series) |
|  | { |
|  | sw.Write(p.X.ToString()); |
|  | sw.Write(","); |
|  | sw.WriteLine(p.Y.ToString()); |
|  | } |
|  | } |
|  | } |
|  |  |
|  | private void MenuExit\_Click(Object sender, System.EventArgs e) |
|  | { |
|  | Application.Exit(); //終了 |
|  | } |
|  |  |
|  | private void MenuCircuitView\_Click(Object sender, System.EventArgs e) |
|  | { |
|  | AboutBox CircuitView = new AboutBox(); |
|  | CircuitView.ShowDialog(); |
|  | } |
|  |  |
|  | /// <summary>グラフの目盛等を描画するメソッドです。</summary> |
|  | ///<param name="frequencyMin" >描画する周波数の最小値を指定します。</param> |
|  | /// <param name="frequencyMax">描画する周波数の最大値を指定します。</param> |
|  | private void DrawSheet(float frequencyMin, float frequencyMax) |
|  | { |
|  |  |
|  | int i, j; |
|  | int digitMin = 0; |
|  | int digitMax = 0; |
|  |  |
|  | digitMin = (int)(Math.Log10(frequencyMin)); |
|  | digitMax = (int)(Math.Log10(frequencyMax)); |
|  | //範囲を左下(最小周波数, 最小利得-20dB)と右上(最大周波数, 最大利得2dB),  余白の上下を20pixelと左右30pixel,X軸線形軸,Y軸対数軸でグラフを描画 |
|  | myGraph = new Graph(GraphDisplay, frequencyMin, GvRangeMin - 4,  frequencyMax, GvRangeMax + 3, 30, 20, Graph.AxisStyle.XlogYlinear); |
|  |  |
|  | //背景色は白 |
|  | myGraph.BackColor = Color.White; |
|  |  |
|  | //フォントの変更 |
|  | myGraph.FontSet("ＭＳ ゴシック", 10, FontStyle.Regular, Color.Black); |
|  |  |
|  | //線を実線、幅2、黒に変更 |
|  | myGraph.LineSet(DashStyle.Solid, 2, Color.Black); |
|  | myGraph.DrawRectangle(frequencyMin, GvRangeMin, frequencyMax,  GvRangeMax); |
|  |  |
|  | //X軸の目盛線を追加 |
|  | for (i = digitMin; i <= digitMax; i += 1) |
|  | { |
|  | for (j = 1; j <= 9; j += 1) |
|  | { |
|  | if (j == 1) |
|  | { |
|  | myGraph.LineSet(DashStyle.Solid, 1, Color.Green); |
|  | myGraph.DrawText(j \* (float)Math.Pow(10, i) \* 0.9F,  GvRangeMin - 2F, AddPrefixUnit(j \* (float)Math.Pow(10, i))); |
|  | } |
|  | else |
|  | { |
|  | myGraph.LineSet(DashStyle.Dash, 1, Color.Green); |
|  | } |
|  | myGraph.DrawLine(j \* (float)Math.Pow(10, i), GvRangeMin,  j \* (float)Math.Pow(10, i), GvRangeMax); |
|  | } |
|  | } |
|  | //X軸のタイトル表示 |
|  | myGraph.FontStyle = FontStyle.Bold; |
|  | myGraph.DrawText((float)Math.Pow(10, (digitMax - digitMin - 1)),  GvRangeMin - 4, "周波数f[Hz]"); |
|  | myGraph.FontStyle = FontStyle.Regular; |
|  |  |
|  | //Y軸の目盛線を追加 |
|  | for (i = (int)GvRangeMin; i <= (int)GvRangeMax; i += 5) |
|  | { |
|  | if (i == 0) |
|  | { |
|  | myGraph.LineSet(DashStyle.Solid, 1, Color.Green); |
|  | } |
|  | else |
|  | { |
|  | myGraph.LineSet(DashStyle.Dash, 1, Color.Green); |
|  | } |
|  | myGraph.DrawLine(frequencyMin, i, frequencyMax, i); |
|  | myGraph.DrawText(frequencyMin \* 0.7F, i + 0.5F,  AddPrefixUnit(i).PadLeft(4)); |
|  | } |
|  | //Y軸のタイトル表示 |
|  | myGraph.FontStyle = FontStyle.Bold; |
|  | myGraph.DrawText(frequencyMin \* 0.8F, GvRangeMax + 2F, "電圧利得Gv[dB]"); |
|  | myGraph.FontStyle = FontStyle.Regular; |
|  | } |
|  |  |
|  | /// <summary>数値にSI接頭辞を付加して文字列を返すメソッドです。</summary> |
|  | /// <param name="number">数値を指定します。</param> |
|  | /// <returns>小数点以下第3位までの数値にSI接頭辞を文字列を返します。</returns> |
|  | private string AddPrefixUnit(float number) |
|  | { |
|  | return AddPrefixUnit(number,""); |
|  | } |
|  |  |
|  | /// <summary>数値にSI接頭辞と単位を付加して文字列を返すメソッドです。</summary> |
|  | /// <param name="number">数値を指定します。</param> |
|  | /// <param name="unit">単位を指定します。省略した場合は単位は付加されません。  </param> |
|  | /// <returns>小数点以下第3位までの数値にSI接頭辞を文字列を返します。</returns> |
|  | private string AddPrefixUnit(float number, string unit) |
|  | { |
|  | string addNumber = string.Empty; |
|  |  |
|  | if (number > Math.Pow(10, 12)) |
|  | addNumber = (number / Math.Pow(10, 12)).ToString("0.###") + "T"; |
|  | else if (number > Math.Pow(10, 9)) |
|  | addNumber = (number / Math.Pow(10, 9)).ToString("0.###") + "G"; |
|  | else if (number > Math.Pow(10, 6)) |
|  | addNumber = (number / Math.Pow(10, 6)).ToString("0.###") + "M"; |
|  | else if (number > Math.Pow(10, 3)) |
|  | addNumber = (number / Math.Pow(10, 3)).ToString("0.###") + "k"; |
|  | else if (number > Math.Pow(10, 0)) |
|  | addNumber = number.ToString("0.###"); |
|  | else if (number > Math.Pow(10, -3)) |
|  | addNumber = (number \* Math.Pow(10, 3)).ToString("0.###") + "m"; |
|  | else if (number > Math.Pow(10, -6)) |
|  | addNumber = (number \* Math.Pow(10, 6)).ToString("0.###") + "u"; |
|  | else if (number > Math.Pow(10, -9)) |
|  | addNumber = (number \* Math.Pow(10, 9)).ToString("0.###") + "n"; |
|  | else if (number > Math.Pow(10, -12)) |
|  | addNumber = (number \* Math.Pow(10, 12)).ToString("0.###") + "p"; |
|  | else |
|  | addNumber = number.ToString("0.###"); |
|  | return addNumber + unit; |
|  | } |
|  | } |
|  | } |

解答例のソースコード(ハイパスフィルタのクラス)

|  |  |
| --- | --- |
| 行 | プログラム |
|  | using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using System.Text; |
|  |  |
|  | namespace Filter |
|  | { |
|  | using System; |
|  | using System.Collections; |
|  | using System.Collections.Generic; |
|  | using System.Data; |
|  | using System.Diagnostics; |
|  |  |
|  | /// <summary>抵抗とコンデンサの一次フィルタ用のクラスです。</summary> |
|  | /// <remarks>各種パラメータ及びプロパティはDecimal型を適用しています。</remarks> |
|  | public class HighPassFilter |
|  | { |
|  | //プロパティ及びメソッド用メンバー |
|  | private decimal rValue; //抵抗値 |
|  | private decimal cValue; //静電容量 |
|  |  |
|  | /// <summary>ハイパスフィルタのインスタンスを生成します。</summary> |
|  | /// <param name="r">抵抗値を指定します。</param> |
|  | /// <param name="c">コンデンサの静電容量を指定します。</param> |
|  | /// <remarks>インスタンス生成時に抵抗とコンデンサの値でフィルタの特性が  設定されます。</remarks> |
|  | public HighPassFilter(decimal r, decimal c) |
|  | { |
|  | //抵抗とコンデンサの値 |
|  | rValue = r; |
|  | cValue = c; |
|  | } |
|  |  |
|  | /// <summary>フィルタの入力信号（電圧と周波数を指定）から  電圧利得を取得するメソッドです。</summary> |
|  | /// <param name="inputVoltage">入力電圧を指定します。</param> |
|  | /// <param name="inputFrequency">入力周波数(Hz)を指定します。</param> |
|  | /// <returns>電圧利得を返します。</returns> |
|  | /// <remarks>抵抗とコンデンサの値はインスタンス生成時の値が  適用されます。</remarks> |
|  | public decimal GetGain(decimal inputVoltage, decimal inputFrequency) |
|  | { |
|  | //出力電圧と角周波数ωとωCR算出用の変数 |
|  | decimal vout,gv , w , wcr; |
|  | //角周波数ω算出 |
|  | w = 2 \* (decimal)Math.PI \* inputFrequency; |
|  | //ωCR算出 |
|  | wcr = w \* cValue \* rValue; |
|  | //電圧利得の算出 |
|  | vout = (decimal)(Math.Sqrt(Math.Pow((double)wcr, 2) /  (1 + Math.Pow((double)wcr, 2))) \* (double)inputVoltage); |
|  | gv = 20 \* (decimal)(Math.Log10((double)(vout / inputVoltage))); |
|  | //電圧利得を返す |
|  | return gv; |
|  | } |
|  |  |
|  | /// <summary>フィルタのカットオフ周波数を取得するメソッドです。</summary> |
|  | /// <returns>カットオフ周波数を返します。</returns> |
|  | /// <remarks>抵抗とコンデンサの値はインスタンス生成時の値が適用されます。</remarks> |
|  | public decimal GetCutoffFrequency() |
|  | { |
|  | decimal fc; |
|  |  |
|  | fc = (1 / (2 \* (decimal)Math.PI \* rValue \* cValue)); |
|  |  |
|  | return fc; |
|  | } |
|  |  |
|  | /// <summary>このフィルタの抵抗値を取得します。</summary> |
|  | /// <returns>抵抗値を単位[Ω]で返します。</returns> |
|  | /// <remarks>このプロパティは確認用です。</remarks> |
|  | public decimal Rohm |
|  | { |
|  | get { return rValue; } |
|  | } |
|  |  |
|  | /// <summary>このフィルタの静電容量を取得します。</summary> |
|  | /// <returns>静電容量を単位[F]で返します。</returns> |
|  | /// <remarks>このプロパティは確認用です。</remarks> |
|  | public decimal Cfarad |
|  | { |
|  | get { return cValue; } |
|  | } |
|  | } |
|  | } |