```
1  //Kunal Mukherjee
2  //3/2/2019
3  //EE 380: Filter design
4
5  using System;
6  using System.Windows;
7  using System.Windows.Media.Imaging;
8  using System.Windows.Forms.DataVisualization.Charting;
9
10 namespace Project2_Filters
11 {
12     /// <summary>
13     /// Interaction logic for MainWindow.xaml
14     /// </summary>
15     public partial class MainWindow : Window
16     {
17         public MainWindow()
18         {
19             InitializeComponent();
20             txtbxR.Text = "0";
21             txtbxC.Text = "0";
22             txtbxL.Text = "0";
23
24         }
25
26         //initializing variables
27         private double r = 0;
28         private double c = 0;
29         private double l = 0;
30         private double wc = 0;
31         private double filter = 0;
32         private double Mh = 0; //magnitude of H
33         private double Ah = 0; //angle of H
34         private double w0 = 0; //undamped natural frequency
35         private double wl = 0; //lower cut-off frequency
36         private double wu = 0; //upper cut-off frequency
37         private double B = 0; //bandwidth
38         private double Q = 0; //quality factor
39         private double w = 0;
40         private double tempVal = 0;
41         private int nI = 0;
42         private int nMax = 0;
43         private System.Windows.Controls.Image img = new
             System.Windows.Controls.Image();
44         public BitmapImage bmi;
45         public WriteableBitmap wbm;
46         public Series wASeries = new Series();
47         public Series wPSeries = new Series();
48         private Chart chtAmp = new Chart();
```

```
49          private Chart chtPha = new Chart();
50
51          //The function loads the |H| chart
52          private void cnvChartAmp_Loaded(object sender, RoutedEventArgs e)
53          {
54              System.Windows.Forms.Integration.WindowsFormsHost host =
55                      new System.Windows.Forms.Integration.WindowsFormsHost();
56              host.Child = chtAmp;
57              // Add the chart to the canvas so it can be displayed.
58              this.cnvChartAmp.Children.Add(host);
59          }
60
61          //The function loads the <H chart
62          private void cnvChartPha_Loaded(object sender, RoutedEventArgs e)
63          {
64              System.Windows.Forms.Integration.WindowsFormsHost host =
65                      new System.Windows.Forms.Integration.WindowsFormsHost();
66              host.Child = chtPha;
67              // Add the chart to the canvas so it can be displayed.
68              this.cnvChartPha.Children.Add(host);
69          }
70
71          //The function displays the appriopiate transfer function and the
              circut
72          private void btnShow_Click(object sender, RoutedEventArgs e)
73          {
74              imgDrawing.Source = wbm;
75
76              if (filter == 0)
77              {
78                  lblHN.Content = wc.ToString("E3");
79                  lblHD.Content = "s + " + wc.ToString("E3");
80              }
81              if (filter == 1)
82              {
83                  lblHN.Content = "s";
84                  lblHD.Content = "s + " + wc.ToString("E3");
85              }
86              if (filter == 2)
87              {
88                  lblHN.Content = " " + B.ToString("E3") + "s";
89                  lblHD.Content = "s^2 + " + B.ToString("E3") + "s + " +
                      (Math.Pow(w0, 2)).ToString("E3");
90              }
91              if (filter == 3)
92              {
93                  lblHN.Content = "s^2 + " + (Math.Pow(w0, 2)).ToString("E2");
94                  lblHD.Content = "s^2 + " + B.ToString("E3") + "s + " +
                      (Math.Pow(w0, 2)).ToString("E2");
```

```
95              }
96          }
97
98          //The function resets the R,L,C value to zero as well as the filter
              values
99          private void btnReset_Click(object sender, RoutedEventArgs e)
100         {
101             txtbxR.Text = "0";
102             txtbxC.Text = "0";
103             txtbxL.Text = "0";
104             txtbxCF.Text = "0";
105             txtbxHCF.Text = "0";
106             txtbxLCF.Text = "0";
107
108             lblHD.Content = "D(s)";
109             lblHN.Content = "H(s)";
110         }
111
112         //The function selects which canvas to display the necessary options
113         private void CmbFilterType_DropDownClosed(object sender, EventArgs e)
114         {
115             if ((cmbFilterType.Text.Equals("Low-Pass")) ||
116                (cmbFilterType.Text.Equals("High-Pass")))
117             {
118                 cnvBP.Width = 164;
119                 cnvLP.Width = 0;
120             }
121             else
122             {
123                 cnvBP.Width = 0;
124                 cnvLP.Width = 164;
125             }
126
127         }
128
129         //The function recalculates the series acording to the zooming option
130         //so that the x-axis can be zoomed in accordingly
131         private void btnZoomIn_Click(object sender, RoutedEventArgs e)
132         {
133             double centFreq = 0;
134
135             if (filter == 1 || filter == 2)
136             {
137                 centFreq = wc;
138             }
139             else
140             {
141                 centFreq = w0;
142             }
```

```
143
144             chtAmp.ChartAreas[0].AxisX.Maximum = centFreq + (0.95 *
                  (chtAmp.ChartAreas[0].AxisX.Maximum - centFreq));
145             chtAmp.ChartAreas[0].AxisX.Minimum = centFreq - (0.95 * (centFreq -
                   chtAmp.ChartAreas[0].AxisX.Minimum));
146
147             chtPha.ChartAreas[0].AxisX.Maximum = centFreq + (0.95 *
                  (chtAmp.ChartAreas[0].AxisX.Maximum - centFreq));
148             chtPha.ChartAreas[0].AxisX.Minimum = centFreq - (0.95 * (centFreq -
                   chtAmp.ChartAreas[0].AxisX.Minimum));
149
150             chtAmp.Series.Clear();
151             chtAmp.Series.Add(wASeries);
152             chtAmp.ChartAreas[0].AxisX.IsLogarithmic = true;
153             chtAmp.ChartAreas[0].AxisX.LogarithmBase = 10;
154             chtAmp.ChartAreas[0].AxisX.Title = "Frequency";
155             chtAmp.ChartAreas[0].AxisX.LabelStyle.Format = "E3";
156             chtAmp.ChartAreas[0].AxisY.Title = "|H|";
157             chtAmp.ChartAreas[0].AxisY.LabelStyle.Format = "{0.00}";
158
159             chtPha.Series.Clear();
160             chtPha.Series.Add(wPSeries);
161             chtPha.ChartAreas[0].AxisX.IsLogarithmic = true;
162             chtPha.ChartAreas[0].AxisX.LogarithmBase = 10;
163             chtPha.ChartAreas[0].AxisX.Title = "Frequency";
164             chtPha.ChartAreas[0].AxisX.LabelStyle.Format = "E3";
165             chtPha.ChartAreas[0].AxisY.Title = "<H";
166             chtPha.ChartAreas[0].AxisY.LabelStyle.Format = "{0.00}";
167         }
168
169         //The function calls the appriopiate transfer function depending on the
                filter type
170         //the function also get the R,L,C value as well as load the correct
171         //curcit image
172         private void btnPlot_Click(object sender, RoutedEventArgs e)
173         {
174             //select the filter type
175             if (cmbFilterType.Text.Equals("Low-Pass"))
176             {
177                 filter = 0;
178             }
179             else if (cmbFilterType.Text.Equals("High-Pass"))
180             {
181                 filter = 1;
182             }
183             else if (cmbFilterType.Text.Equals("Band-Pass"))
184             {
185                 filter = 2;
186             }
```

```
187             else if (cmbFilterType.Text.Equals("Band-Reject"))
188             {
189                 filter = 3;
190             }
191 
192             //get the r,c,l value
193             if (txtbxR.Text != "")
194             {
195                 r = getRLCValue(0); // Convert.ToDouble(txtbxR.Text);
196             }
197             else
198             {
199                 r = 10;
200                 txtbxR.Text = "10";
201             }
202             if (txtbxC.Text != "")
203             {
204                 c = getRLCValue(1); //Convert.ToDouble(txtbxC.Text);
205             }
206             else
207             {
208                 c = 10;
209                 txtbxC.Text = "10";
210             }
211             if (txtbxL.Text != "")
212             {
213                 l = getRLCValue(2); //Convert.ToDouble(txtbxL.Text);
214             }
215             else
216             {
217                 l = 10;
218                 txtbxL.Text = "10";
219             }
220 
221 
222             //depending on the choice select the filter type
223             if (filter == 0)
224             {
225                 if (c == 0)
226                 {
227                     plotLowPassRLFilter();
228 
229                     bmi = new BitmapImage(new Uri(@"C:\Users\kunmu\Documents
                        \Kunal\UE courses\EE-380\Project2_Filters\Project2_Filters
                        \Image\LPRL.png"));
230                     wbm = new WriteableBitmap(bmi);
231 
232                 }
233                 else
```

```
234                 {
235                     plotLowPassRCFilter();
236 
237                     bmi = new BitmapImage(new Uri(@"C:\Users\kunmu\Documents
                        \Kunal\UE courses\EE-380\Project2_Filters\Project2_Filters
                        \Image\LPRC.png"));
238                     wbm = new WriteableBitmap(bmi);
239                 }
240             }
241             if (filter == 1)
242             {
243                 if (c == 0)
244                 {
245                     plotHighPassRLFilter();
246 
247                     bmi = new BitmapImage(new Uri(@"C:\Users\kunmu\Documents
                        \Kunal\UE courses\EE-380\Project2_Filters\Project2_Filters
                        \Image\HPRL.png"));
248                     wbm = new WriteableBitmap(bmi);
249 
250                 }
251                 else
252                 {
253                     plotHighPassRCFilter();
254 
255                     bmi = new BitmapImage(new Uri(@"C:\Users\kunmu\Documents
                        \Kunal\UE courses\EE-380\Project2_Filters\Project2_Filters
                        \Image\HPRC.png"));
256                     wbm = new WriteableBitmap(bmi);
257                 }
258             }
259             if (filter == 2)
260             {
261                 plotBandPassFilter();
262 
263                 bmi = new BitmapImage(new Uri(@"C:\Users\kunmu\Documents\Kunal
                    \UE courses\EE-380\Project2_Filters\Project2_Filters\Image
                    \BP.png"));
264                 wbm = new WriteableBitmap(bmi);
265 
266             }
267             if (filter == 3)
268             {
269                 plotBandRejectFilter();
270 
271                 bmi = new BitmapImage(new Uri(@"C:\Users\kunmu\Documents\Kunal
                    \UE courses\EE-380\Project2_Filters\Project2_Filters\Image
                    \BR.png"));
272                 wbm = new WriteableBitmap(bmi);
```

```
273             }
274 
275         }
276 
277         //The function allows u-micro, p-pico, n-nano, f-femto, m-mili,k,K-kilo
               and M-mega
278         //character to be used for R,L,C values
279         private double getRLCValue(int choice)
280         {
281             string num = "";
282             if (choice == 0)
283             {
284                 if (txtbxR.Text.Contains("K") || txtbxR.Text.Contains("k"))
285                 {
286                     num = txtbxR.Text.Substring(0, txtbxR.Text.Length - 1);
287                     return Convert.ToDouble(num) * Math.Pow(10, 3);
288                 }
289                 if (txtbxR.Text.Contains("M"))
290                 {
291                     num = txtbxR.Text.Substring(0, txtbxR.Text.Length - 1);
292                     return Convert.ToDouble(num) * Math.Pow(10, 6);
293                 }
294                 if (txtbxR.Text.Contains("m"))
295                 {
296                     num = txtbxR.Text.Substring(0, txtbxR.Text.Length - 1);
297                     return Convert.ToDouble(num) * Math.Pow(10, -3);
298                 }
299                 if (txtbxR.Text.Contains("u"))
300                 {
301                     num = txtbxR.Text.Substring(0, txtbxR.Text.Length - 1);
302                     return Convert.ToDouble(num) * Math.Pow(10, -6);
303                 }
304                 if (txtbxR.Text.Contains("n"))
305                 {
306                     num = txtbxR.Text.Substring(0, txtbxR.Text.Length - 1);
307                     return Convert.ToDouble(num) * Math.Pow(10, -9);
308                 }
309                 if (txtbxR.Text.Contains("p"))
310                 {
311                     num = txtbxR.Text.Substring(0, txtbxR.Text.Length - 1);
312                     return Convert.ToDouble(num) * Math.Pow(10, -12);
313                 }
314                 if (txtbxR.Text.Contains("f"))
315                 {
316                     num = txtbxR.Text.Substring(0, txtbxR.Text.Length - 1);
317                     return Convert.ToDouble(num) * Math.Pow(10, -15);
318                 }
319                 else
320                 {
```

```
321                         return Convert.ToDouble(txtbxR.Text);
322                     }
323                 }
324                 if(choice == 1)
325                 {
326                     if (txtbxC.Text.Contains("M"))
327                     {
328                         num = txtbxC.Text.Substring(0, txtbxC.Text.Length - 1);
329                         return Convert.ToDouble(num) * Math.Pow(10, 6);
330                     }
331                     if (txtbxC.Text.Contains("K") || txtbxC.Text.Contains("k"))
332                     {
333                         num = txtbxC.Text.Substring(0, txtbxC.Text.Length - 1);
334                         return Convert.ToDouble(num) * Math.Pow(10, 3);
335                     }
336                     if (txtbxC.Text.Contains("m"))
337                     {
338                         num = txtbxC.Text.Substring(0, txtbxC.Text.Length - 1);
339                         return Convert.ToDouble(num) * Math.Pow(10, -3);
340                     }
341                     if (txtbxC.Text.Contains("u"))
342                     {
343                         num = txtbxC.Text.Substring(0, txtbxC.Text.Length - 1);
344                         return Convert.ToDouble(num) * Math.Pow(10, -6);
345                     }
346                     if (txtbxC.Text.Contains("n"))
347                     {
348                         num = txtbxC.Text.Substring(0, txtbxC.Text.Length - 1);
349                         return Convert.ToDouble(num) * Math.Pow(10, -9);
350                     }
351                     if (txtbxC.Text.Contains("p"))
352                     {
353                         num = txtbxC.Text.Substring(0, txtbxC.Text.Length - 1);
354                         return Convert.ToDouble(num) * Math.Pow(10, -12);
355                     }
356                     if (txtbxC.Text.Contains("n"))
357                     {
358                         num = txtbxC.Text.Substring(0, txtbxC.Text.Length - 1);
359                         return Convert.ToDouble(num) * Math.Pow(10, -15);
360                     }
361                     else
362                     {
363                         return Convert.ToDouble(txtbxC.Text);
364                     }
365                 }
366                 if(choice == 2)
367                 {
368                     if (txtbxL.Text.Contains("M"))
369                     {
```

```
370                     num = txtbxL.Text.Substring(0, txtbxL.Text.Length - 1);
371                     Console.WriteLine(num);
372                     return Convert.ToDouble(num) * Math.Pow(10, 6);
373                 }
374                 if (txtbxL.Text.Contains("K") || txtbxL.Text.Contains("k"))
375                 {
376                     num = txtbxL.Text.Substring(0, txtbxL.Text.Length - 1);
377                     Console.WriteLine(num);
378                     return Convert.ToDouble(num) * Math.Pow(10, 3);
379                 }
380                 if (txtbxL.Text.Contains("m"))
381                 {
382                     num = txtbxL.Text.Substring(0, txtbxL.Text.Length - 1);
383                     Console.WriteLine(num);
384                     return Convert.ToDouble(num) * Math.Pow(10, -3);
385                 }
386                 if (txtbxL.Text.Contains("u"))
387                 {
388                     num = txtbxL.Text.Substring(0, txtbxL.Text.Length - 1);
389                     Console.WriteLine(num);
390                     return Convert.ToDouble(num) * Math.Pow(10, -6);
391                 }
392                 if (txtbxL.Text.Contains("n"))
393                 {
394                     num = txtbxL.Text.Substring(0, txtbxL.Text.Length - 1);
395                     Console.WriteLine(num);
396                     return Convert.ToDouble(num) * Math.Pow(10, -9);
397                 }
398                 if (txtbxL.Text.Contains("p"))
399                 {
400                     num = txtbxL.Text.Substring(0, txtbxL.Text.Length - 1);
401                     Console.WriteLine(num);
402                     return Convert.ToDouble(num) * Math.Pow(10, -12);
403                 }
404                 if (txtbxL.Text.Contains("f"))
405                 {
406                     num = txtbxL.Text.Substring(0, txtbxL.Text.Length - 1);
407                     Console.WriteLine(num);
408                     return Convert.ToDouble(num) * Math.Pow(10, -15);
409                 }
410                 else
411                 {
412                     return Convert.ToDouble(txtbxL.Text);
413                 }
414             }
415 
416             return 0;
417         }
418 
```

```
419         //The function calculates the number of sig.fig for a number given
420         private int magnitudeQuantificationofValue(double number)
421         {
422             int counter = 0;
423 
424             if (number > 1)
425             {
426                 while (number > 10)
427                 {
428                     number /= 10;
429                     counter++;
430                 }
431             }
432 
433             return counter;
434         }
435 
436         //The function maximum and the minimum w value for the transfer
              function
437         //for low and high pass filter
438         private void setWCAxisMinandMax(double wTemp)
439         {
440             int x = magnitudeQuantificationofValue(wTemp);
441 
442             chtAmp.ChartAreas[0].AxisX.Minimum = Math.Pow(10, x - 2);
443             chtPha.ChartAreas[0].AxisX.Minimum = Math.Pow(10, x - 2);
444 
445             chtAmp.ChartAreas[0].AxisX.Maximum = Math.Pow(10, x + 2);
446             chtPha.ChartAreas[0].AxisX.Maximum = Math.Pow(10, x + 2);
447 
448             nI = magnitudeQuantificationofValue(chtAmp.ChartAreas
                  [0].AxisX.Minimum);
449             nMax = magnitudeQuantificationofValue(chtAmp.ChartAreas
                  [0].AxisX.Maximum);
450         }
451 
452         //The function maximum and the minimum w value for the transfer
              function
453         //for band pass and stop filter
454         private void setW0AxisMinandMax()
455         {
456             if (txtbxLCF.Text == "0")
457             {
458                 w0 = 1 / Math.Sqrt(l * c);
459                 B = r / l;
460 
461                 double rTemp = Math.Sqrt(Math.Pow((B / 2), 2) + Math.Pow(w0,
                      2));
462 
```

```
463                 wl = (-(B / 2) + rTemp);
464                 wu = (+(B / 2) + rTemp);
465 
466                 txtbxLCF.Text = (wl).ToString("E3");
467                 txtbxHCF.Text = (wu).ToString("E3");
468 
469             }
470             else
471             {
472                 wl = Convert.ToDouble(txtbxLCF.Text);
473                 wu = Convert.ToDouble(txtbxHCF.Text);
474 
475                 w0 = Math.Sqrt(wl * wu);
476                 B = wu - wl;
477 
478                 if (txtbxR.Text != "0")
479                 {
480                     txtbxL.Text = (getRLCValue(0) / B).ToString( "E3");
481                     txtbxC.Text = (1 / (getRLCValue(2) * Math.Pow(w0,
                        2))).ToString("E3"); ;
482                 }
483                 else if (txtbxC.Text != "0")
484                 {
485                     txtbxL.Text = (1 / Math.Pow(w0, 2) * getRLCValue
                        (1)).ToString("E3");
486                     txtbxR.Text = (getRLCValue(2) * B).ToString( "E3");
487                 }
488                 else
489                 {
490                     txtbxC.Text = (1 / Math.Pow(w0, 2) * getRLCValue
                        (2)).ToString("E3");
491                     txtbxR.Text = (getRLCValue(2) * B).ToString( "E3");
492                 }
493 
494             }
495 
496             int xMin = magnitudeQuantificationofValue(wl);
497             int xMax = magnitudeQuantificationofValue(wu);
498 
499             chtAmp.ChartAreas[0].AxisX.Minimum = Math.Pow(10, xMin - 3);
500             chtPha.ChartAreas[0].AxisX.Minimum = Math.Pow(10, xMin - 3);
501 
502             chtAmp.ChartAreas[0].AxisX.Maximum = Math.Pow(10, xMax + 3);
503             chtPha.ChartAreas[0].AxisX.Maximum = Math.Pow(10, xMax + 3);
504 
505             nI = magnitudeQuantificationofValue(chtAmp.ChartAreas
                  [0].AxisX.Minimum);
506             nMax = magnitudeQuantificationofValue(chtAmp.ChartAreas
                  [0].AxisX.Maximum);
```

```
507
508         }
509
510         //The function clears the series up and sets it up for data population
511         private void initializeSeries()
512         {
513             //clear the chart area
514             chtAmp.ChartAreas.Clear();
515             chtAmp.ChartAreas.Add("Default");
516             chtPha.ChartAreas.Clear();
517             chtPha.ChartAreas.Add("Default");
518
519             chtAmp.Width = 450;
520             chtAmp.Height = 270;
521             chtPha.Width = 450;
522             chtPha.Height = 270;
523
524             chtAmp.Location = new System.Drawing.Point(0, 0);
525             chtPha.Location = new System.Drawing.Point(0, 0);
526
527             //w for |H|
528             wASeries = new Series();
529             //w for <H
530             wPSeries = new Series();
531
532             wASeries.ChartType = SeriesChartType.Line;
533             wPSeries.ChartType = SeriesChartType.Line;
534         }
535
536         //The function fills up the graph after series is calculated
537         //as well as formats the graph
538         private void fillSeriesGraph()
539         {
540             chtAmp.Series.Clear();
541             chtAmp.Series.Add(wASeries);
542             chtAmp.ChartAreas[0].AxisX.IsLogarithmic = true;
543             chtAmp.ChartAreas[0].AxisX.LogarithmBase = 10;
544             chtAmp.ChartAreas[0].AxisX.Title = "Frequency";
545             chtAmp.ChartAreas[0].AxisX.LabelStyle.Format = "E3";
546             chtAmp.ChartAreas[0].AxisY.Title = "|H|";
547             chtAmp.ChartAreas[0].AxisY.LabelStyle.Format = "{0.00}";
548
549
550
551             chtPha.Series.Clear();
552             chtPha.Series.Add(wPSeries);
553             chtPha.ChartAreas[0].AxisX.IsLogarithmic = true;
554             chtPha.ChartAreas[0].AxisX.LogarithmBase = 10;
555             chtPha.ChartAreas[0].AxisX.Title = "Frequency";
```

```
556             chtPha.ChartAreas[0].AxisX.LabelStyle.Format = "E3";
557             chtPha.ChartAreas[0].AxisY.Title = "<H";
558             chtPha.ChartAreas[0].AxisY.LabelStyle.Format = "{0.00}";

559         }
560
561         //The function calculates the transfer function for
562         //low pass RL filter
563         private void plotLowPassRLFilter()
564         {
565             initializeSeries();
566
567             //calculate the cutoff frequency
568             wc = r / l;
569
570             if (txtbxCF.Text == "0")
571             {
572                 txtbxCF.Text = wc.ToString("E3");
573             }
574             else
575             {
576                 wc = Convert.ToDouble(txtbxCF.Text);
577
578                 if(txtbxR.Text == "0")
579                 {
580                     txtbxR.Text = (wc * getRLCValue(2)).ToString("E3");
581                 }
582                 else
583                 {
584                     txtbxL.Text = (getRLCValue(0) / wc).ToString("E3");
585                 }
586             }
587
588             setWCAxisMinandMax(wc);
589
590             //generate the graph points |H| and <H
591             for (int n = nI; n <= nMax; n++)
592             {
593                 for (int m = 1; m < 10; m++)
594                 {
595                     w = m * Math.Pow(10, n);
596
597                     Mh = 20 * Math.Log10(1 / Math.Sqrt(1 + Math.Pow((w / wc),
                        2)));
598                     Ah = (-Math.Atan2(w, wc)) * (180 / Math.PI);
599
600                     wASeries.Points.AddXY(w, Mh);
601                     wPSeries.Points.AddXY(w, Ah);
602
```

```
603                 }
604
605             }
606
607             fillSeriesGraph();
608         }
609
610         //The function calculates the transfer function for
611         //low pass RC filter
612         private void plotLowPassRCFilter()
613         {
614             initializeSeries();
615
616             wc = 1 / (r * c);
617
618             if (txtbxCF.Text == "0")
619             {
620                 txtbxCF.Text = wc.ToString("E3");
621             }
622             else
623             {
624                 wc = Convert.ToDouble(txtbxCF.Text);
625
626                 if (txtbxR.Text == "0")
627                 {
628                     txtbxR.Text = (1 / wc * getRLCValue(1)).ToString("E3");
629                 }
630                 else
631                 {
632                     txtbxC.Text = (1 / wc * getRLCValue(0)).ToString("E3");
633                 }
634             }
635
636             setWCAxisMinandMax(wc);
637
638             //generate the graph points |H| and <H
639             for (int n = nI; n <= nMax; n++)
640             {
641                 for (int m = 1; m < 10; m++)
642                 {
643                     w = m * Math.Pow(10, n);
644
645                     Mh = 20 * Math.Log10(1 / Math.Sqrt(1 + Math.Pow((w / wc),
                        2)));
646                     Ah = (-Math.Atan2(w, wc)) * (180 / Math.PI);
647
648                     wASeries.Points.AddXY(w, Mh);
649                     wPSeries.Points.AddXY(w, Ah);
```

```
650                 }
651             }
652
653             fillSeriesGraph();
654         }
655
656         //The function calculates the transfer function for
657         //high pass RL filter
658         private void plotHighPassRLFilter()
659         {
660             initializeSeries();
661
662             wc = r / l;
663
664             if (txtbxCF.Text == "0")
665             {
666                 txtbxCF.Text = wc.ToString("E3");
667             }
668             else
669             {
670                 wc = Convert.ToDouble(txtbxCF.Text);
671
672                 if (txtbxR.Text == "0")
673                 {
674                     txtbxR.Text = (wc * getRLCValue(2)).ToString("E3");
675                 }
676                 else
677                 {
678                     txtbxL.Text = (getRLCValue(0) / wc).ToString("E3");
679                 }
680             }
681
682             setWCAxisMinandMax(wc);
683
684             //generate the graph points |H| and <H
685             for (int n = nI; n <= nMax; n++)
686             {
687                 for (int m = 1; m < 10; m++)
688                 {
689                     w = m * Math.Pow(10, n);
690
691                     Mh = 20 * Math.Log10(1 / Math.Sqrt(1 + Math.Pow((wc / w),
                        2)));
692                     Ah = (Math.Atan2(wc, w)) * (180 / Math.PI);

693
694                     wASeries.Points.AddXY(w, Mh);
695                     wPSeries.Points.AddXY(w, Ah);
696                 }
```

```
697             }
698
699             fillSeriesGraph();
700         }
701
702         //The function calculates the transfer function for
703         //high pass RC filter
704         private void plotHighPassRCFilter()
705         {
706             initializeSeries();
707
708             wc = 1 / (r * c);
709
710             if (txtbxCF.Text == "0")
711             {
712                 txtbxCF.Text = wc.ToString( "E3");
713             }
714             else
715             {
716                 wc = Convert.ToDouble(txtbxCF.Text);
717
718                 if (txtbxR.Text == "0")
719                 {
720                     txtbxR.Text = (1 / wc * getRLCValue(1)).ToString( "E3");
721                 }
722                 else
723                 {
724                     txtbxC.Text = (1 / wc * getRLCValue(0)).ToString( "E3");
725                 }
726             }
727
728             setWCAxisMinandMax(wc);
729
730             //generate the graph points |H| and <H
731             for (int n = nI; n <= nMax; n++)
732             {
733                 for (int m = 1; m < 10; m++)
734                 {
735                     w = m * Math.Pow(10, n);
736
737                     Mh = 20 * Math.Log10(1 / Math.Sqrt(1 + Math.Pow((wc / w),
                        2)));
738                     Ah = (Math.Atan2(wc, w)) * (180 / Math.PI);
739
740                     wASeries.Points.AddXY(w, Mh);
741                     wPSeries.Points.AddXY(w, Ah);
742                 }
743
```

```
744             }
745
746             fillSeriesGraph();
747         }
748
749         //The function calculates the transfer function for
750         //band pass filter
751         private void plotBandPassFilter()
752         {
753             initializeSeries();
754
755             setW0AxisMinandMax();
756
757             for (int n = nI; n <= nMax; n++)
758             {
759                 for (int m = 1; m < 10; m++)
760                 {
761                     w = m * Math.Pow(10, n);
762
763                     tempVal = (Math.Pow(w0, 2)) - (Math.Pow(w, 2));
764
765                     Mh = 20 * Math.Log10(Math.Sqrt(Math.Pow(B * w, 2)) /
                    Math.Sqrt(Math.Pow(tempVal, 2) + Math.Pow(B * w, 2)));
766                     Ah = (Math.Atan2(B * w, 0) - Math.Atan2(B * w, tempVal)) *
                    (180 / Math.PI);
767
768                     wASeries.Points.AddXY(w, Mh);
769                     wPSeries.Points.AddXY(w, Ah);
770                 }
771
772             }
773
774             fillSeriesGraph();
775         }
776
777         //The function calculates the transfer function for
778         //band stop filter
779         private void plotBandRejectFilter()
780         {
781             initializeSeries();
782
783             setW0AxisMinandMax();
784
785             for (int n = nI; n <= nMax; n++)
786             {
787                 for (int m = 1; m < 10; m++)
788                 {
789                     w = m * Math.Pow(10, n);
790
```

```
791                     tempVal = (Math.Pow(w0, 2)) - (Math.Pow(w, 2));
792
793                     if (Math.Sqrt(Math.Pow(tempVal, 2))  != 0)
794                     {
795                         Mh = 20 * Math.Log10(Math.Sqrt(Math.Pow(tempVal, 2)) /
                        Math.Sqrt(Math.Pow(tempVal, 2) + Math.Pow(B * w, 2)));
796                     }
797
798                     Ah = (Math.Atan2(0, tempVal) - Math.Atan2(B * w, tempVal))
                        * (180 / Math.PI);
799
800                     wASeries.Points.AddXY(w, Mh);
801                     wPSeries.Points.AddXY(w, Ah);
802                 }
803
804             }
805
806             fillSeriesGraph();
807         }
808
809     }
810 }
811
```