Hans Guthrie
5/12/2014
ECE 540
Dr. Day

Project V: Fourier Based System Modeling

State Variable Modeling of a Feedback Control System

$$H(s) = \frac{K_v * K_p * s + K_v * K_i}{J * s^3 + s^2 + K_v * K_p * s + K_v * K_i}$$

a) We used the above transfer equation to simulate the system using the following parameters: `dt = 0.1ms, J = 100ms, Kv = 5.0 (rad/sec/volt), the gains defined in Table 1.`

K_p	K_i
4	0
8	0
12	0
4	2
8	2
12	2
4	4
8	4
12	4

Table 1 – Set of Gains to be Simulated

Next, to visualize the results we made 6 plots. The first 3 hold K_p constant and let K_i move. Table 2 shows a list of the plots. The plots are included in Appendix 2.

Plot #	K_p Values	K_i Values
1	1	0, 2, 3
2	2	0, 4, 6
3	4	0, 8, 12
4	1, 2, 4	0*K_p
5	1, 2, 4	2*K_p
6	1, 2, 4	3*K_p

Table 2 – Set of plots

b) We then repeated this for a more realistic saw tooth wave. Table 3 shows the plots.

Plot #	K_p Values	K_i Values	Period
7	1, 2, 4	2*K_p	4
8	1, 2, 4	2*K_p	8

Table 3 – Set of plots (continued)

c) See Appendix 1 for the time lag calculations.

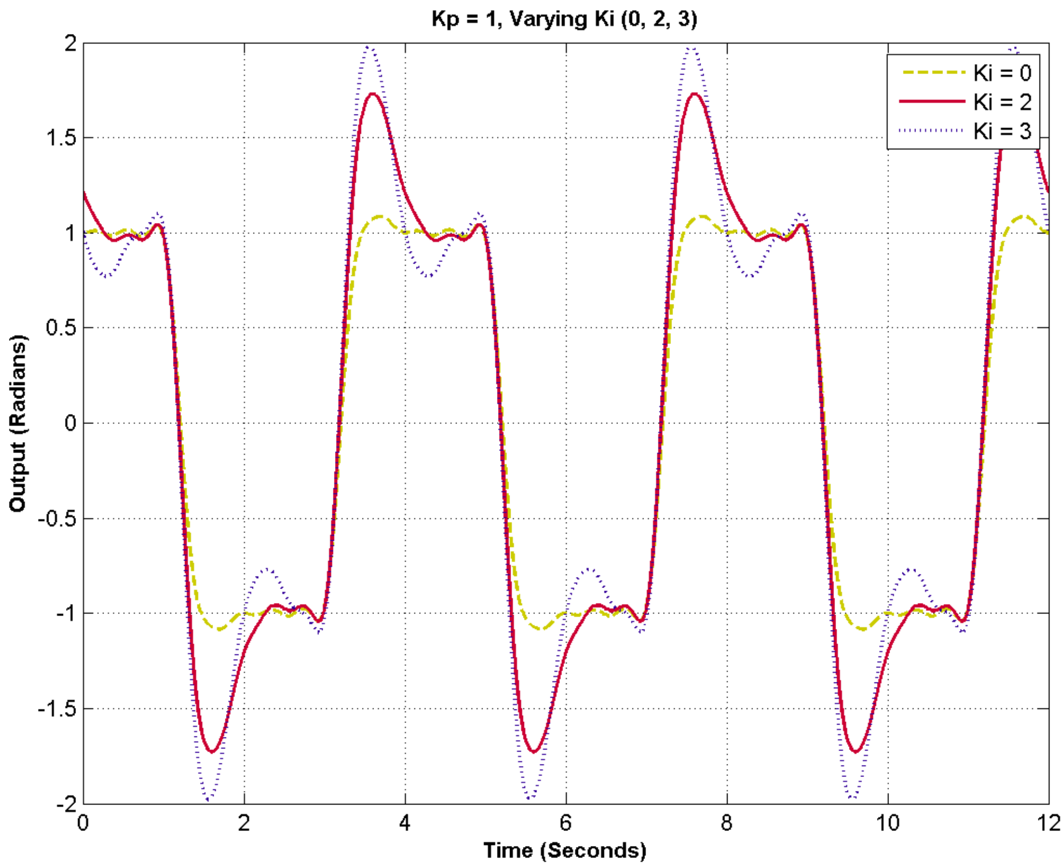
d) See Appendix 1 for the time lag calculations. The time lag changed when we increased the period to 8 seconds.

Appendix 1: Program Output

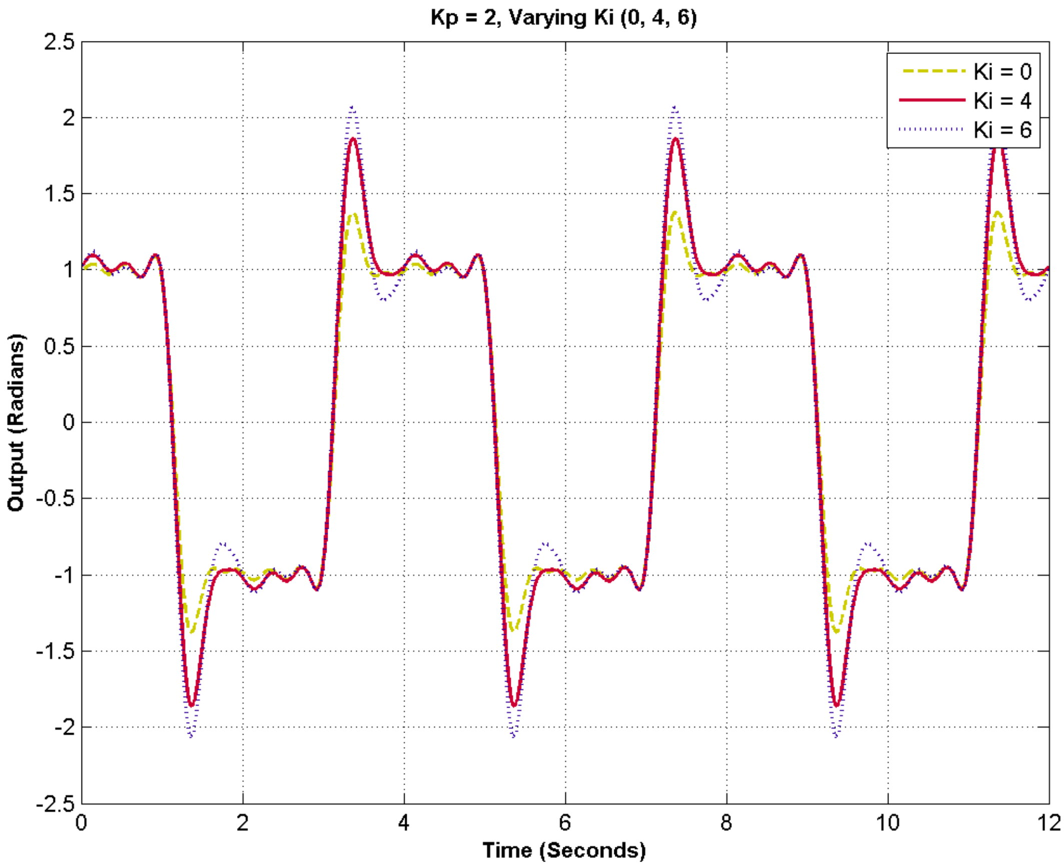
```
Calculating Part a.
100% Complete
Success!

Calculating Part b with period = 4.
Time Lag: 0.1460 seconds for Kp = 1, Ki = 2.
Time Lag: 0.1138 seconds for Kp = 2, Ki = 4.
Time Lag: 0.0845 seconds for Kp = 4, Ki = 8.
Success!

Calculating Part b with period = 8.
Time Lag: 0.1808 seconds for Kp = 1, Ki = 2.
Time Lag: 0.1179 seconds for Kp = 2, Ki = 4.
Time Lag: 0.0614 seconds for Kp = 4, Ki = 8.
Success!

Press any key to quit.
```

Appendix 2: Plots

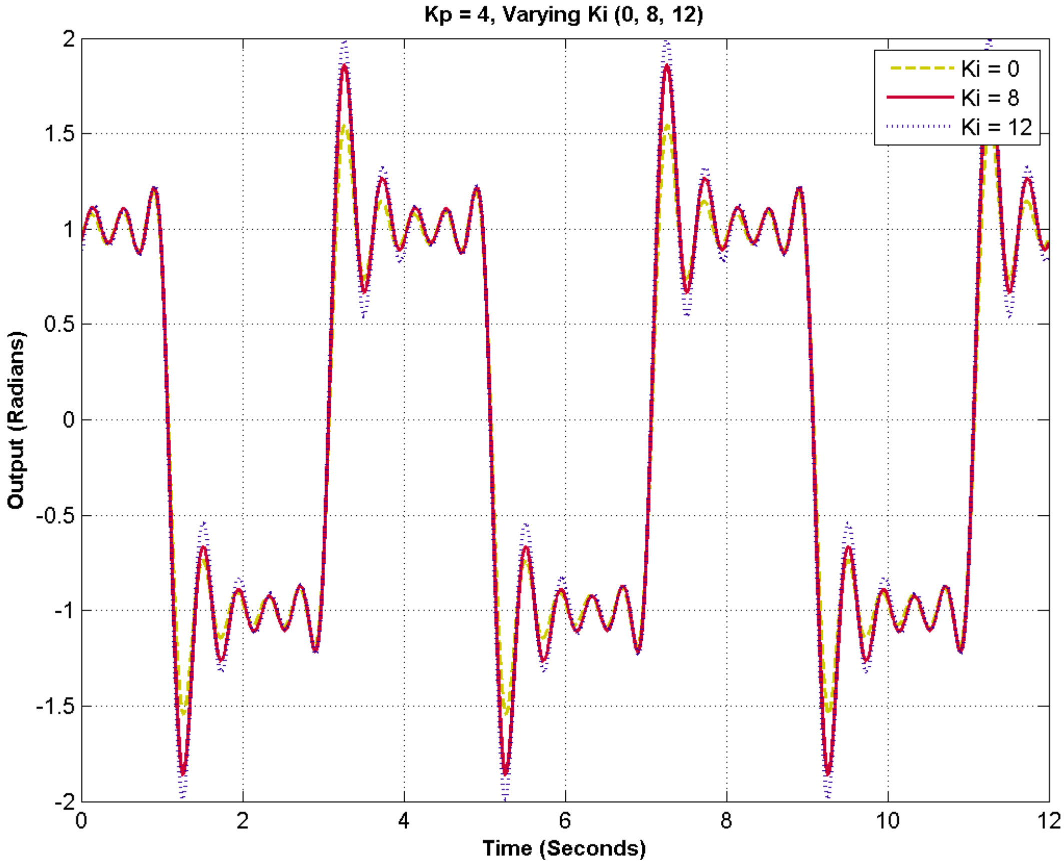
Plot 1



Plot 2

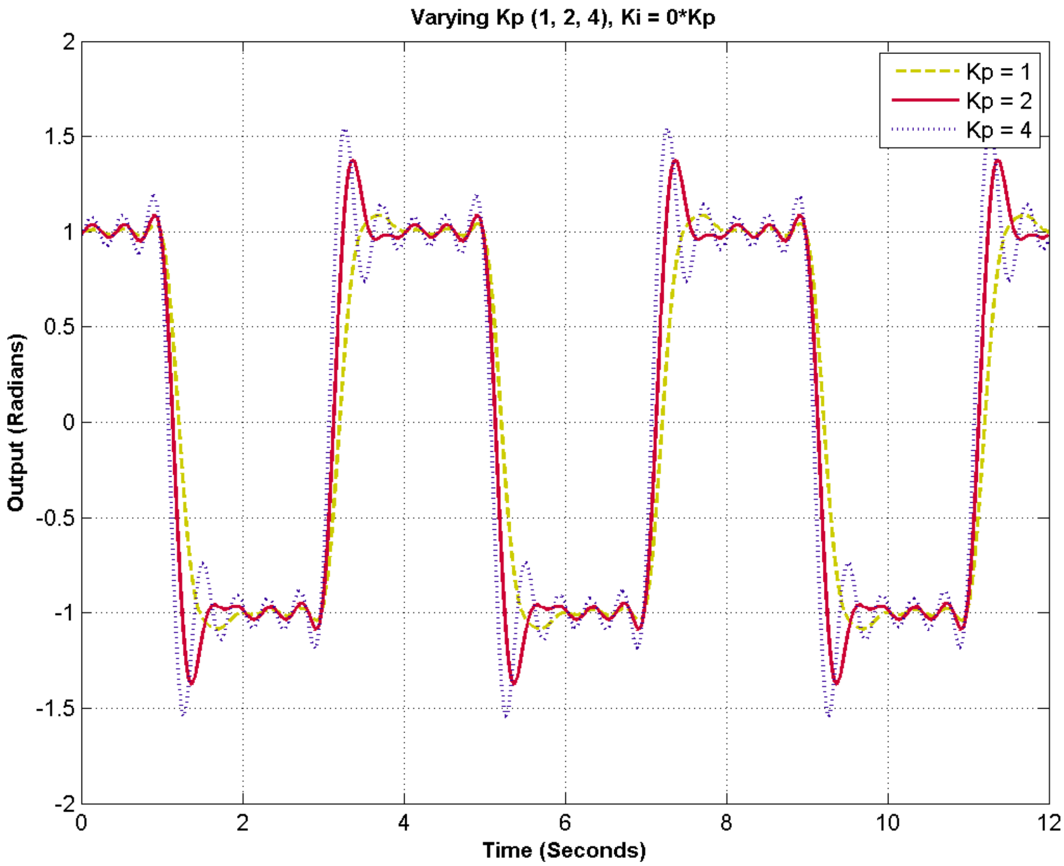

Kp = 2, Varying Ki (0, 4, 6)
Ki = 0
Ki = 4
Ki = 6
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
Output (Radians)
0
2
4
6
8
10
12
Time (Seconds)

Plot 3



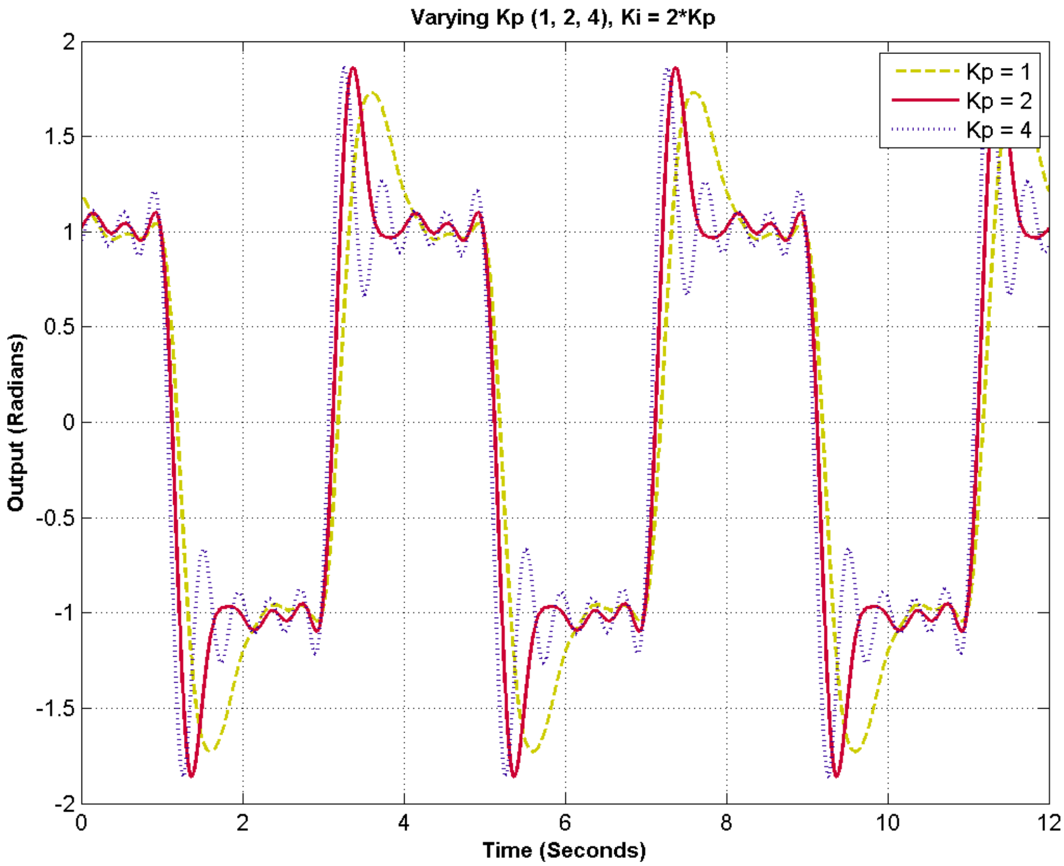
Kp = 4, Varying Ki (0, 8, 12)
Ki = 0
Ki = 8
Ki = 12
Output (Radians)
Time (Seconds)
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
2
4
6
8
10
12

Plot 4



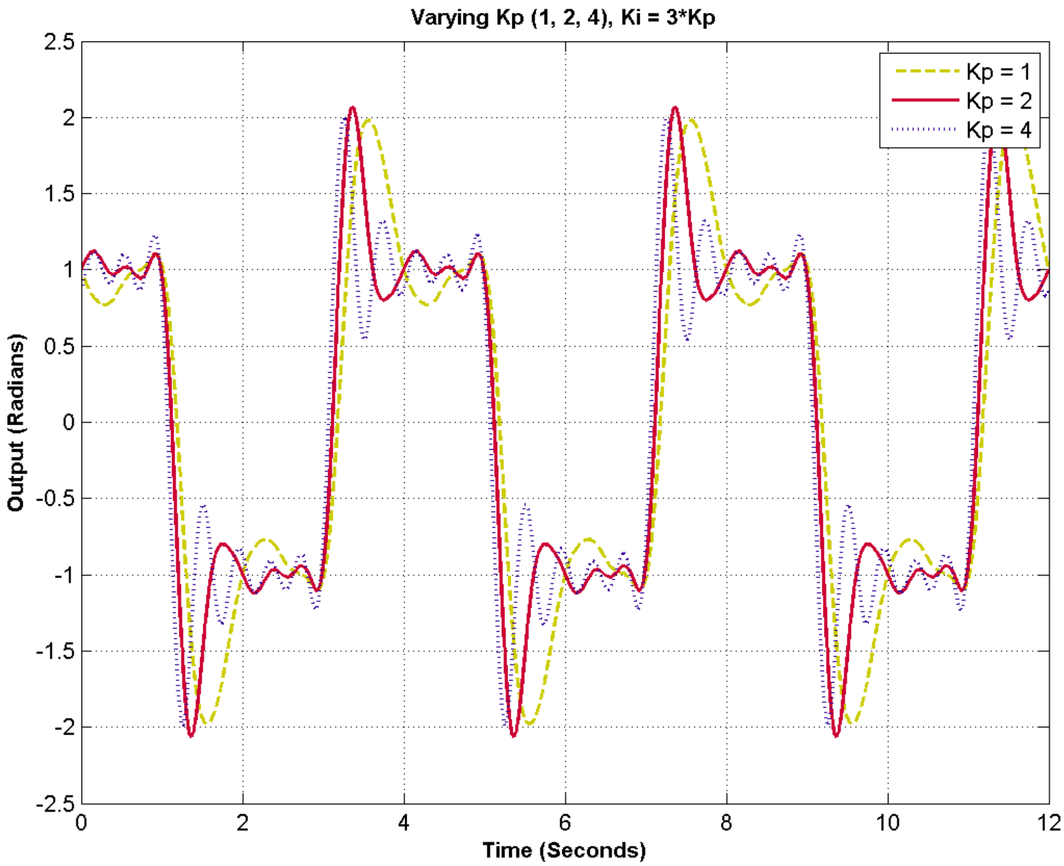
Varying Kp (1, 2, 4), Ki = 0*Kp
Kp = 1
Kp = 2
Kp = 4
Output (Radians)
Time (Seconds)
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
2
4
6
8
10
12

Plot 5



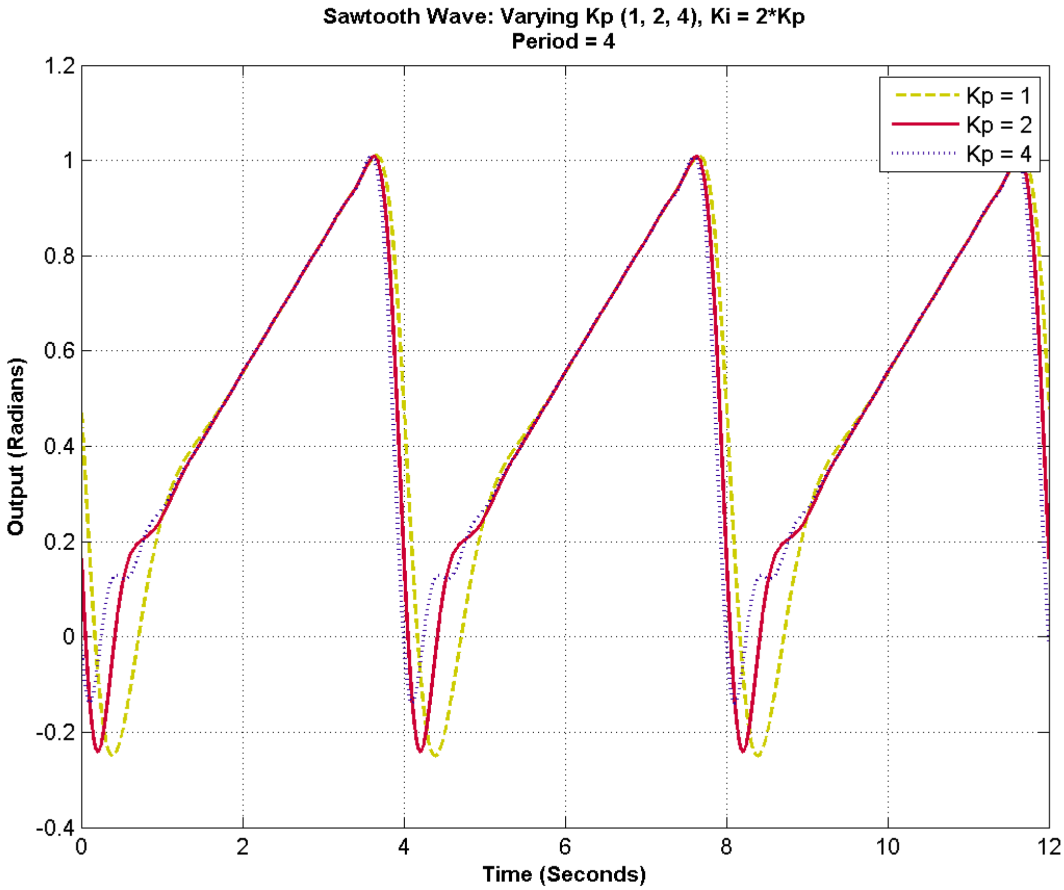
Varying Kp (1, 2, 4), Ki = 2*Kp
Kp = 1
Kp = 2
Kp = 4
Output (Radians)
Time (Seconds)
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
2
4
6
8
10
12

Plot 6

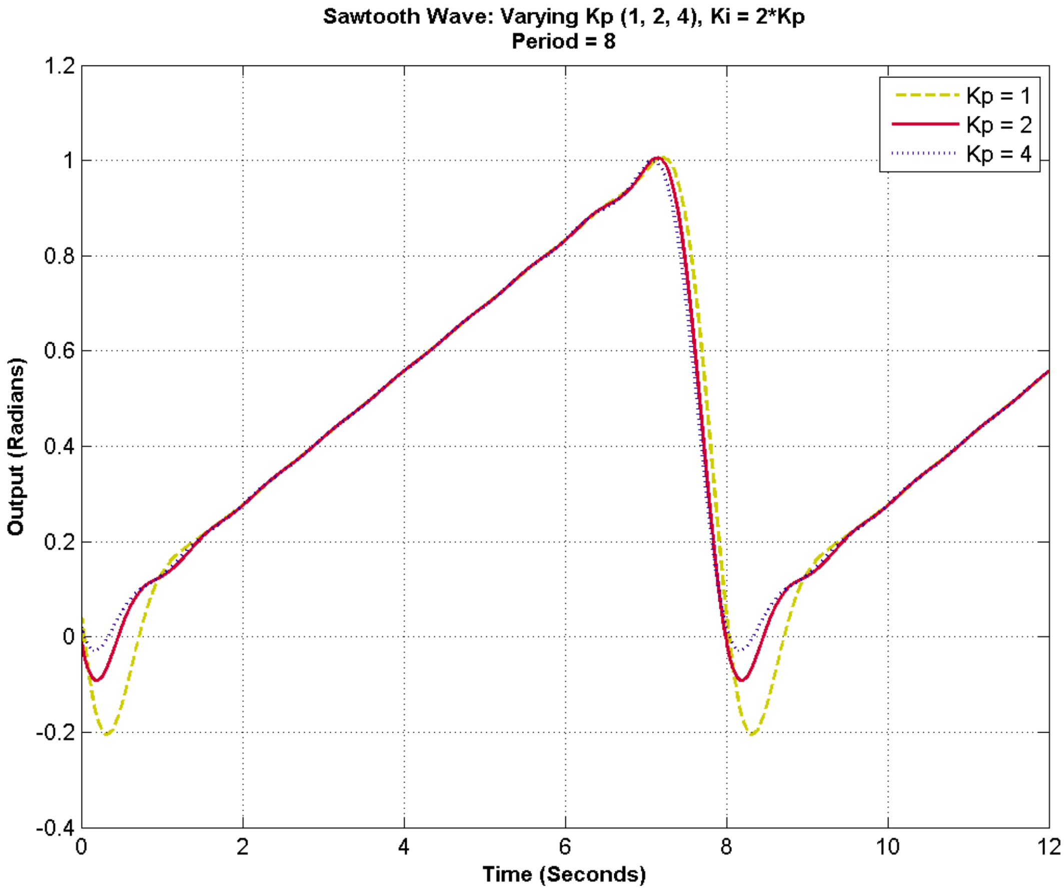


Varying Kp (1, 2, 4), Ki = 3*Kp
Kp = 1
Kp = 2
Kp = 4
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
Output (Radians)
0
2
4
6
8
10
12
Time (Seconds)

Plot 7


Sawtooth Wave: Varying Kp (1, 2, 4), Ki = 2*Kp
Period = 4
Kp = 1
Kp = 2
Kp = 4
Output (Radians)
Time (Seconds)
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
0
2
4
6
8
10
12

Plot 8



Sawtooth Wave: Varying Kp (1, 2, 4), Ki = 2*Kp
Period = 8
Kp = 1
Kp = 2
Kp = 4
Output (Radians)
Time (Seconds)
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
0
2
4
6
8
10
12

Appendix 3: Code

```
/* Hans Guthrie
 * ECE 540
 * Project 5
 * Note: I worked with Nelson Padgett on this project
 */

#pragma warning( disable:4996 )
#include <math.h>
#include <stdio.h>
#include <complex>

using namespace std;

#define eps 2.22e-16
int main( );
void part_a( );
void part_b( int T );
double CalculateTimeLag( complex<double> *in, complex<double> *out, int T );

// Support Routine for complex class.
// Calculates the magnitide of a
double mag( complex<double> a )
{
	double r, i;
	r = a.real( );
	i = a.imag( );
	return( sqrt( r*r + i*i ) );
}

// Support Routine for complex class.
// Calculates the complex conjugate of a
complex<double> conjg( complex<double> a )
{
	double r, i;
	r = a.real( );
	i = a.imag( );
	return( complex<double>( r, -i ) );
}

// Writes the array to the output file as a CSV
void WriteToFile( char *name, complex<double> *out, int length, double Dt )
{
	FILE *fout = fopen( name, "w" );

	double T = 0.0;

	for ( int k = 0; k < length; k++ )
		fprintf( fout, "%18.16lg,%18.16lg,%18.16lg\n", out[ k ].real( ), out[ k
].imag( ), T = T + Dt );
	fclose( fout );
}

// Program to simulate the steady state response of
// a circuit to a square wave.
#define SimulationSteps 120000
```

```
// Part a of the project
void part_a( )
{
	double w,
		T = 4.0, // Period set at 4.0 seconds.
		Dt = 0.1e-3, // Step length (seconds)
		J = 100e-3,
		Kv = 5.0,
		PI = 4.0*atan( 1.0 ),
		progressbarcounter = 0.0;

	complex<double> Hjkw, Xk, Outw, s;
	complex<double> *in = new complex<double>[ SimulationSteps ]; // Array of complex
to hold inputs.
	complex<double> *out = new complex<double>[ SimulationSteps ]; // Array of complex
to hold outputs.

	int k, m, Ki, Kp, i;
	char name[ 32 ];

	printf( "Calculating Part a.\n" );

	FILE *fout = fopen( "FourierCoefficients.csv", "w" ); // the file we're reading in
from

	w = 2 * PI / T;

	//Dt = 3.0*T / SimulationSteps; // Compute step size to generate 3 periods.

	//loop through all the Kp's -- but only 1, 2, 4. (not 3)
	for ( Kp = 1; Kp <= 4; Kp *= 2 )
	{
		//loop through the Ki's
		for ( i = 0; i <= 2; i++ )
		{
			if ( i == 0 )
			{
				Ki = 0;
			}
			else
			{
				Ki = Kp*( 1 + i );
			}

			// Initialize output as zero.
			for ( m = 0; m < SimulationSteps; m++ )
			{
				out[ m ] = 0.0;
				in[ m ] = 0.0;
			}

			// Generate the output
			k = 1;
			while ( k < 10 )
			{
				if ( k ) // Compute Fourier Coefficients for square wave.
				{
					Xk = 1.0 * sin( k*PI / 2 ) / ( k*PI / 2 );
```

```
                                                }

                                                // Compute Filter response at this frequency
                                                s = complex<double>( 0.0, k*w );
                                                Hjkw = ( Kv*Kp*s + Kv*Ki ) / ( J*s*s*s + s*s + Kv*Kp*s + Kv*Ki
);

                                                // Add in this term into steady state response.
                                                for ( m = 0; m < SimulationSteps; m++ )
                                                {
                                                        // Create the value of exp(j*t*w) for frequency (k*w)
and time (m*Dt)
                                                        Outw = exp( complex<double>( 0.0, m*Dt*k*w ) );

                                                        // Compute Fourier Series representation of Input.
                                                        in[ m ] = in[ m ] + Xk * Outw;
                                                        // Note k = 0 is a special case
                                                        if ( k ) // which is not a conjugate pair.
                                                                in[ m ] = in[ m ] + conjg( Xk * Outw );

                                                        // Compute Fourier Series representation of Output.
                                                        //out[ m ] = out[ m ] + Xk * Hjkw * Outw;
                                                        out[ m ] += Xk * Hjkw * Outw;
                                                        // Note k = 0 is a special case
                                                        if ( k ) // which is not a conjugate pair.
                                                                out[ m ] += conjg( Xk * Hjkw * Outw );
                                                }// End of Loop through time steps.

                                                //Create/update progress bar - developed by Nelson and I
                                                progressbarcounter += 1.0;
                                                printf( "%3.0f%% Complete\r", ( progressbarcounter / 45.0 ) *
100 );

                                                if ( k >= 9 )
                                                {
                                                        sprintf( name, "FS_output_Kp_%d_Ki_%d.csv", Kp, Ki );
                                                        WriteToFile( name, out, SimulationSteps, Dt );
                                                }

                                                k += 2; //increment k for next iteration
                                        }// End of loop through Fourier Series Compnoents.
                                }
                        }
        printf( "\nSuccess!\n\n" );
}// End of part_a

//Part b of the project
void part_b( int T )
{
        double w,
                Dt = 0.1e-3,
                J = 100e-3,
                Kv = 5,
                PI = 4.0*atan( 1.0 ),
                progressbarcounter = 0.0;

        complex<double> Hjkw, Xk, Outw, w1, w2, s;
```

```
	complex<double> *in = new complex<double>[ SimulationSteps ]; // Array of complex 
to hold inputs.
	complex<double> *out = new complex<double>[ SimulationSteps ]; // Array of complex 
to hold outputs.

	printf( "Calculating Part b with period = %d.\n", T );

	int m, Kp, Ki, n;
	char name[ 32 ];
	FILE *fout = fopen( "FourierCoefficients2.csv", "w" );
	w = 2 * PI / T;

	//loop through the Kp's
	for ( Kp = 1; Kp <= 4; Kp = 2 * Kp )
	{
		Ki = 2 * Kp;
		//initialize out and in to 0's
		for ( m = 0; m < SimulationSteps; m++ )
		{
			out[ m ] = 0.0;
			in[ m ] = 0.0;
		}
		// Generate input and output

		int k = 0;
		while ( k < 10 )
		{
			if ( k ) // Compute Fourier Coefficient k for sawtooth wave.
			{
				w1 = complex<double>( 0.0, 0.2*k*PI );
				w2 = complex<double>( 0.0, 1.8*k*PI );
				Xk = ( 1 / 0.9 )*( -10.0 + 9.0*( 1.0 - w1 )*exp( w1 ) + ( 1.0 
+ w2 )*exp( -w2 ) ) / ( 4.0*k*k*PI*PI );
			}
			else // k == 0...
			{
				Xk = 0.5;
			}
			// Compute Filter response at this frequency
			s = complex<double>( 0.0, k*w );
			Hjkw = ( Kv*Kp*s + Kv*Ki ) / ( J*s*s*s + s*s + Kv*Kp*s + Kv*Ki );
			fprintf( fout, "%18.16lg, %18.16lg, %18.16lg, %18.16lg\n", Xk.real( 
), Xk.imag( ), Hjkw.real( ), Hjkw.imag( ) );

			// Loop through time.
			for ( m = 0; m < SimulationSteps; m++ )
			{
				// Create the value of exp( j*t*w ) for this

				// frequency (k*w) and time (m*Dt);
				Outw = exp( complex<double>( 0.0, m*Dt*k*w ) );

				// Compute Fourier Series representation of Input.
				in[ m ] = in[ m ] + Xk * Outw;

				// Note k = 0 is a special case
				if ( k ) // which is not a conjugate pair.
				{
```

```
                        in[ m ] = in[ m ] + conjg( Xk * Outw );
                    }

                    // Compute Fourier Series representation of Output.
                    out[ m ] = out[ m ] + Xk * Hjkw * Outw;

                    // Note k = 0 is a special case
                    if ( k ) // which is not a conjugate pair.
                    {
                        out[ m ] = out[ m ] + conjg( Xk * Hjkw * Outw );
                    }
                } // End of time loop.

                //displays progress percentage - developed by Nelson and I
                progressbarcounter += 1.0;
                printf( "%3.0f%% Complete\r", ( progressbarcounter / 30.0 ) * 100 );

                if ( k >= 9 )
                {
                    // Save off input and output model at this point.
                    sprintf( name, "FS_RST_output_Kp_%d_Ki_%d_T_%d.csv", Kp, Ki, T
);
                    WriteToFile( name, out, SimulationSteps, Dt );
                    //Print time lab
                    double Timelag = CalculateTimeLag( in, out, T );
                    printf( "Time Lag: %.4f seconds for Kp = %d, Ki = %d.\n",
Timelag, Kp, Ki );
                }
                k++;
            }
        }
        printf( "Success!\n\n" );
}//end of part_b

//return and then print from the part_b method
double CalculateTimeLag( complex<double> *in, complex<double> *out, int T )
{
        double MaxInput = 0.0,
               MaxOutput = 0.0,
               TimeLag,
               Steps;
        int time1, time2;

        //Prevent it from using multiple period's data
        if ( T == 4 )
        {
               Steps = SimulationSteps / 2;
        }
        else
        {
               Steps = SimulationSteps;
        }
        for ( int i = 0; i < Steps; i++ )
        {
               if ( out[ i ].real( ) > MaxOutput )
               {
                       MaxOutput = out[ i ].real( );
                       time1 = i;
```

```
		}

		if ( in[ i ].real( ) > MaxInput )
		{
			MaxInput = in[ i ].real( );
			time2 = i;
		}
	}

	//Calculate the timelag
	TimeLag = abs( time2 - time1 );
	TimeLag *= ( 0.1e-3 );
	return TimeLag;
	//move to part_b method since I'm returning a double
	//printf( "Time Lag is %lg Seconds for Kp = %i and Ki = %i.\n", TimeLag, Kp, Ki );
}

int main( )
{
	part_a( ); //part a

	part_b( 4 ); //Part b with a time period of 4 seconds
	part_b( 8 ); //Part b with a time period of 8 seconds

	printf( "Press any key to quit." );
	getchar( );
}
```

```
a) Square wave Simulation.
      Code ......................  6/6
      Scale Correct .............  6/6
      Plots .....................  8/8

b) Sawtooth
      Simulation ................   4/4
      Scale .....................   3/3
      Plots .....................   3/3

c) Time Shift for 4 second period
      Valid Search ..............   6/6
      Correct Values ............   4/4

e) Time shift for 8 second period
      Search/Values .............   6/6
      Did they change ...........   4/4

Total .........................  50/50
```