Physics 121 – Problem Set # 2

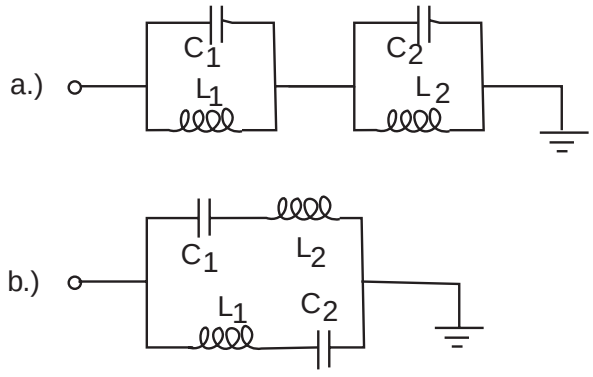
(due Friday, April 19)

1. Compute the Fourier transform $\tilde{f}(\omega)$ of the following functions:

$$\begin{aligned} a.) \qquad f(t) &= e^{-|t|/a} \\ b.) \qquad f(t) &= \begin{cases} 1 & -a/2 < t < a/2 \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{1}$$

   For each case, use contour integration to invert the transform and reconstruct $f(t)$ from $\tilde{f}(\omega)$.

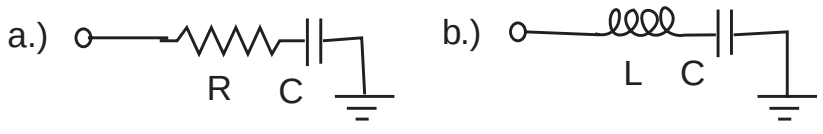
2. Analyze the following circuits. Each one is supposed to be driven by an AC voltage $V_0 \cos \omega t$. Find the effective impedance $Z(\omega)$ and sketch a plot of $1/|Z(\omega)|$. A resonant frequency ω_* is a frequency for which $Z(\omega_*) = 0$. Find the resonant frequencies for each case.



3. Each of the simple circuits below is driven by a voltage

$$V(t) = \begin{cases} 0 & t < 0 \\ 2V_0 t & 0 < t < T/2 \\ 2V_0(T-t) & T/2 < t < T \\ 0 & T < t \end{cases} \tag{2}$$

   Find the current $I(t)$ that flows in response, using the Green's function method.



4. This problem involves a Java applet that computes Fourier transforms numerically.

   (a) To begin, work out the formulae for the discrete approximation to the Fourier transform. We approximate a signal $f(t)$ by a discrete set of values $f(n)$, $n = 0, 1, ..., (N-1)$. In this program, I take $N = 256$. Then let

   $$\tilde{f}(m) = \sum_{n=0}^{n=N-1} f(n) e^{-2\pi i m n/N} \tag{3}$$

   Show that the inversion formula is

   $$f(n) = \frac{1}{N} \sum_{m=0}^{m=N-1} \tilde{f}(m) e^{2\pi i m n/N} \tag{4}$$

   Take the real and imaginary parts of (1) and show that the complex entries $\tilde{f}(m)$ can be represented by real entries $\tilde{f}_{cos}(m)$, $\tilde{f}_{sin}(m)$, $m = 0, 1, ..., N/2$, computed by replacing the exponential in (1) by $\cos(2\pi mn/N)$ or $\sin(2\pi mn/N)$. (Note that $\tilde{f}_{sin}(m) = 0$ for $m = 0$ and $m = N/2$. Write the formula for reconstruction of $f(n)$ from $\tilde{f}_{cos}(m)$ and $\tilde{f}_{sin}(m)$.

   (b) Download the files `FourierLab.html`, `FourierLab.java`, `FourierLabGUI.java`, `PhysicsApplet.java`, and `FourierTransform.java` from the Physics 121 web site `http://www.slac/stanford.edu/~mpeskin/Physics121/java/`. Edit the file `FourierLab.html` to remove the statement `archive = "FourierLab.jar"`. Compile the FourierLab applet and note the following behavior: (1) The applet draws a waveform in the upper box. This waveform is provided by the `inputWave` function in the `FourierLab` code. By pushing the button 'Draw f', you can draw over this function and supply an arbitrary curve. (2) No activity is visible in the middle box. Actually, there is a function supplied by the `inputFilter` function in the `FourierLab` code, but it is set equal to 1 everywhere. By pushing the button 'Draw filter', you can draw over this function. (3) The button 'Finish drawing' turns off both drawing modes. (4) If you press the button 'Transform', nothing happens.

   (c) Look at the code of the `FourierTransform` class. The variables N and $N/2$ are called `Nx` and `N2` in this class. The class defines three vectors with double precision entries. The first, called `f`, has 256 entries (`f[0]` to `f[Nx-1]`). This represents the real-time signal. The second, `fcos`, indexed from 0 to `N2`, represents the cosine series coefficients from part (a). The third, `fsin`, indexed from 0 to `N2`, represents the sine series coefficients from part (a). The function `Transform` is supposed to use the values of `f[n]` to fill the arrays `fcos[m]`, `fsin[m]`. The function `InverseTransform` is supposed to use the values of `fcos[m]`, `fsin[m]` to fill the array `f[n]`. Both functions actually read out zero. Modify these functions to correctly implement the Fourier transforms by carrying out the sums indicated in part (a). Remember that $\cos\theta$, $\sin\theta$, and π appear in java as `Math.cos(theta)`, `Math.sin(theta)`, and `Math.PI`.

(d) Set the filter equal to 1 and transform some illustrative functions. Try a Gaussian wavepacket, a square wave (constant segments of V_0 and $-V_0$), a triangular wave as in Problem 2, and a noisy function that you can obtain by drawing with the mouse. What is plotted in the second box is $|\texttt{fcos}^2[m] + \texttt{fsin}^2[m]|^{1/2}$. The third box plots the result of transforming back to `f[n]`. If the filter equals 1 and the result in the third box is not identical to the curve in the first box, there is an error in your code.

(e) Set the filter equal to

$$1/(1 + (m/60)^2) \ , \tag{5}$$

low-pass filter. The Fourier transform shown in magenta is now multiplied by the filter function. The inverse of the filtered transform appears in the lower box. Try this filter on the Fourier transforms of the illustrative functions in (d). Is the result sensible?

(f) Modify the filter so that it is a high-pass filter. Try this filter on the Fourier transforms of the illustrative functions in (d). Is the result sensible?

(g) We will use the `FourierTransform` class in future problem sets, so you might want to make it more efficient. One simple way to do this it to create a new data members `double[] sins` and `double[] coss` and tabulate the needed values of sines and cosines once and for all. The new arrays can be filled in the constructor `FourierTransform(N)` and can then be used in all transforms without further calls to the trig functions.

This trick reduces the computation of Fourier transforms to simple matrix multiplication. Still, the number of operations is N^2. For serious signal analysis, there is a really souped-up Fourier transform algorithm, the 'Fast Fourier Transform', that requires only $N \log N$ operations.

Hand in your code for the `FourierTransform` class, your solution to (a), and illustrative plots from (d) and (e).

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class FourierLab extends FourierLabGUI{

    void inputWave(){
        for (int n = 0 ; n < Nx; n++){
            double t = (n- 0.5*Nx)/(0.5*Nx);
            F[n] = Math.exp(- 10.0 * t * t) * Math.sin(24.0* Math.PI * t);
        }
    }

    void inputFilter(){
for (int m = 0; m <= Nx; m++){
   /*  Here is an option for no filtering : */
            filter[m] = 1.0;
   /*  Here is a filter that makes a sharp cut in Fourier space: */
        //  filter[m] = 0.0;
//   if (m > 40 && m < 60) filter[m] = 1.0;
   /*  Here is a filter that makes a smooth cutoff of high frequencies:  */
      //    double t = m/60.0;
      //    filter[m] = 1.0/(1 + t*t);
        }
    }

}
```

Figure 1: The source file `FourierLab.java`.

```
public class FourierTransform{

    int Nx, N2;
    double[] f, fcos, fsin;

    FourierTransform(int N){
        Nx = N;
        N2 = Nx/2;
f = new double[Nx];
        fcos = new double[N2+1];
        fsin = new double[N2+1];
    }

    void Transform(){
for (int i = 0; i <= N2 ; i++){
    double Tc = 0.0;
    double Ts = 0.0;
            for (int j = 0; j < Nx ; j++){
/*  do something intelligent here   */
            }
            fcos[i] = Tc;
            fsin[i] = Ts;
}
    }

    void InverseTransform(){
for (int i = 0; i < Nx ; i++){
    double T = fcos[0] + fcos[N2];
            for (int j = 1; j < N2 ; j++){
/*  do something intelligent here   */
            }
            f[i] = 0.0;
}
    }
}
```

Figure 2: The source file `FourierTransform.java`.