University of Pittsburgh
Department of Electrical and Computer Engineering
ECE 0301 – Spring 2021

Assignment #13

Programming concepts:

- Aggregation
- Polymorphism and virtual member functions
- Static and dynamic binding
- Pure virtual member functions
- Abstract base classes

ECE concepts:

- First-order and second-order linear homogenous differential equations with constant coefficients and an initial condition.

Reference: [1] *Advanced Engineering Mathematics, 10th ed.*, Erwin Kreyszig, Wiley, 2011, chapters 1, 2.

Background: For this assignment, you will write software to solve first-order and second-order differential equations that are ordinary, linear and homogeneous, with constant coefficients, and subject to an initial condition. Furthermore, the quantity of interest is a function of time $y(t)$, and all derivatives are with respect to *t*. That's a lot of descriptors, so we will examine each of them so that we know what kind of differential equations will be solved.

The first descriptor for our differential equations is *ordinary*, by which we mean that there is only one independent variable, in this case *t*. In general we could have two or more independent variables, so that the quantity of interest might be $y(t_1, t_2, \ldots t_n)$, and we would have a partial differential equation to solve. General forms for first-order and second-order partial differential equations are:

$$F_1\left(t_1, t_2, \ldots t_n, y, \frac{\partial y}{\partial t_1}, \frac{\partial y}{\partial t_2}, \ldots \frac{\partial y}{\partial t_n}\right) = 0$$

$$F_2\left(t_1, t_2, \ldots t_n, y, \frac{\partial y}{\partial t_1}, \frac{\partial y}{\partial t_2}, \ldots \frac{\partial y}{\partial t_n}, \frac{\partial^2 y}{\partial t_1^2}, \frac{\partial^2 y}{\partial t_2^2}, , \ldots \frac{\partial^2 y}{\partial t_n^2}\right) = 0.$$

Fortunately, the equations we will solve are ordinary, so that the general forms are considerably simpler:

$$F_1\left(t, y, \frac{dy}{dt}\right) = 0$$

$$F_2\left(t, y, \frac{dy}{dt}, \frac{d^2y}{dt^2}\right) = 0.$$

In these equations, the functions F_1 and F_2 can be nonlinear, which can make the equations very challenging to solve. However, the differential equations of interest to us are also *linear*, which means that they are of the form [1]:

$$\frac{dy}{dt} + p(t)y(t) = q(t)$$

$$\frac{d^2y}{dt^2} + p_1(t)\frac{dy}{dt} + p_0(t)y(t) = q(t).$$

Next, the equations we will solve are *homogeneous*, such that $q(t) = 0$:

$$\frac{dy}{dt} + p(t)y(t) = 0$$

$$\frac{d^2y}{dt^2} + p_1(t)\frac{dy}{dt} + p_0(t)y(t) = 0.$$

A further restriction is that the coefficients must be constant in time, so that the final forms for our differential equations are:

$$\frac{dy}{dt} + by(t) = 0$$

$$\frac{d^2y}{dt^2} + a\frac{dy}{dt} + by(t) = 0.$$

Note that the solutions to these equations will involve unspecified constants, one constant for the first-order equation and two for the second-order equation, and in order to determine values for these constants we must have additional information about the system in the form of an *initial condition*. For the first-order equation, we must specify the value of the solution at some particular time, such as $y(t_0) = K_0$, while for the second-order equation we must also specify a value for the derivative at the same time, i.e. $y'(t_0) = K_1$. Therefore, a complete description of the two problems we wish to solve is as follows:

$$\frac{dy}{dt} + by(t) = 0, \quad y(t_0) = K_0$$

$$\frac{d^2y}{dt^2} + a\frac{dy}{dt} + by(t) = 0, \quad y(t_0) = K_0, \quad \left.\frac{dy}{dt}\right|_{t=t_0} = K_1.$$

As we will see, solving for the first-order equation is quite simple, but solving the second-order equation requires considerably more work.

First-Order: If the differential equation is of first-order, and linear and homogeneous, then the problem of interest is

$$\frac{dy}{dt} + by(t) = 0 \tag{1}$$

$$y(t_0) = K_0 \tag{2}$$

We must find a function $y(t)$ that solves (1), subject to the initial condition (2). In this case, we can find the solution through integration [1],

$$\frac{dy}{dt} = -by$$

$$\frac{dy}{y} = -b \cdot dt$$

$$\int \frac{dy}{y} = -\int b \cdot dt$$

$$\ln y = -bt + \ln C$$

$$y(t) = Ce^{-bt}$$

Note that there is an unspecified constant *C* in the solution. What this means is that $y(t)$ is a solution of (1) for any value of *C*, and we need to use our initial condition (2) to determine *C*,

$$y(t_0) = Ce^{-bt_0} = K_0 \quad \Rightarrow \quad C = K_0 e^{bt_0} \quad \Rightarrow \quad y(t) = K_0 e^{bt_0} e^{-bt}.$$

This solution applies for all $t \geq t_0$, and can be simplified to

$$y(t) = K_0 e^{-b(t-t_0)} \quad \text{for} \quad t \geq t_0. \tag{3}$$

Second-order: The second-order differential equation and initial conditions are

$$\frac{d^2y}{dt^2} + a\frac{dy}{dt} + by(t) = 0 \tag{4}$$

$$y(t_0) = K_0 \tag{5}$$

$$\left.\frac{dy}{dt}\right|_{t=t_0} = K_1 \tag{6}$$

We must find a function $y(t)$ that solves (4), subject to the initial conditions (5) and (6). The procedure to find the solution is complicated, because the form of the solution varies depending on the values of *a* and *b*. To determine the proper form, we write the *characteristic equation* for this problem, by replacing each derivative of $y(t)$ with the corresponding power of a variable, say λ,

$$\lambda^2 + a\lambda + b = 0.$$

The roots of this quadratic polynomial determine the form of the solution [1].

- If $a^2 - 4b > 0$, then the roots of the characteristic polynomial are real-valued and distinct, and the solution to (4) is of the form

$$y(t) = c_1 e^{\lambda_1 t} + c_2 e^{\lambda_2 t}.$$

- If $a^2 - 4b = 0$, then the roots of the characteristic polynomial are real-valued and equal, and the solution is of the form
$$y(t) = (c_1 + c_2 t)e^{\lambda t}.$$
- If $a^2 - 4b < 0$, then the roots of the characteristic polynomial are complex-conjugates, and the solution is of the form
$$y(t) = e^{\sigma t}\left(A\cos\omega t + B\sin\omega t\right),$$
where $\sigma = \mathfrak{Re}\{\lambda\}$ and $\omega = \mathfrak{Im}\{\lambda\}$.

Second-order, Case I (Overdamped): When the roots of the characteristic polynomial are real and distinct, the solution is a sum of exponential functions with rates equal to the roots [1],
$$\begin{aligned}\lambda^2 + a\lambda + b &= 0\\ a^2 - 4b &> 0\\ \lambda_1, \lambda_2 &= \frac{-a \pm \sqrt{a^2 - 4b}}{2}\\ y(t) &= c_1 e^{\lambda_1 t} + c_2 e^{\lambda_2 t}.\end{aligned}$$
This is known as the overdamped case, because one of the two exponentials will have a longer time constant than the exponential in the critically damped case. The constants $\{c_1, c_2\}$ must be chosen to satisfy the initial conditions. However, we may use any variables or constant expressions to represent these constants, so let us replace them with
$$c_1 \to c_1 e^{-\lambda_1 t_0} \qquad \text{and} \qquad c_2 \to c_2 e^{-\lambda_2 t_0}.$$
Then the solution for our differential equation becomes
$$\begin{aligned}y(t) &= c_1 e^{-\lambda_1 t_0} e^{\lambda_1 t} + c_2 e^{-\lambda_2 t_0} e^{\lambda_2 t}\\ &= c_1 e^{\lambda_1 (t - t_0)} + c_2 e^{\lambda_2 (t - t_0)}.\end{aligned} \tag{7}$$

We can confirm that (7) is a solution to (4) for any $\{c_1, c_2\}$ by evaluating the derivatives
$$\begin{aligned}\frac{dy}{dt} &= c_1 \lambda_1 e^{\lambda_1 (t-t_0)} + c_1 \lambda_2 e^{\lambda_2 (t-t_0)}\\ \frac{d^2 y}{dt^2} &= c_1 \lambda_1^2 e^{\lambda_1 (t-t_0)} + c_1 \lambda_2^2 e^{\lambda_2 (t-t_0)}\\ \frac{d^2 y}{dt^2} + a\frac{dy}{dt} + by(t) &= c_1\left(\lambda_1^2 + a\lambda_1 + b\right)e^{\lambda_1 (t-t_0)} + c_2\left(\lambda_2^2 + a\lambda_2 + b\right)e^{\lambda_2 (t-t_0)}.\end{aligned}$$
The right-hand side must be zero, because each of the quadratic expressions is zero, since λ_1 and λ_2 are the roots of the characteristic equation. This means that (4) is satisfied.

In our solution, $t - t_0$ appears in place of t in the standard solution. The advantage of this approach is that it simplifies the use of the initial conditions to determine the unspecified constants $\{c_1, c_2\}$,

$$y(t_0) = c_1 e^{\lambda_1(0)} + c_2 e^{\lambda_2(0)} = c_1 + c_2$$
$$\frac{dy}{dt} = c_1\lambda_1 e^{\lambda_1(t-t_0)} + c_2\lambda_2 e^{\lambda_2(t-t_0)}$$
$$\left.\frac{dy}{dt}\right|_{t=t_0} = c_1\lambda_1 + c_2\lambda_2.$$

Therefore, to find $\{c_1, c_2\}$, we must solve a pair of linear equations,

$$c_1 + c_2 = K_0$$
$$c_1\lambda_1 + c_2\lambda_2 = K_1.$$

We can write this as a single matrix-vector equation,

$$\begin{bmatrix} 1 & 1 \\ \lambda_1 & \lambda_2 \end{bmatrix}\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} K_0 \\ K_1 \end{bmatrix},$$

which we will choose to solve through matrix inversion [1],

$$\begin{bmatrix} 1 & 1 \\ \lambda_1 & \lambda_2 \end{bmatrix}^{-1} = \frac{1}{\lambda_2 - \lambda_1}\begin{bmatrix} \lambda_2 & -1 \\ -\lambda_1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ \lambda_1 & \lambda_2 \end{bmatrix}^{-1}\begin{bmatrix} K_0 \\ K_1 \end{bmatrix}$$

$$= \frac{1}{\lambda_2 - \lambda_1}\begin{bmatrix} \lambda_2 & -1 \\ -\lambda_1 & 1 \end{bmatrix}\begin{bmatrix} K_0 \\ K_1 \end{bmatrix}$$

$$= \frac{1}{\lambda_2 - \lambda_1}\begin{bmatrix} K_0\lambda_2 - K_1 \\ -\lambda_1 K_0 + K_1 \end{bmatrix}.$$

This solution will be invalid when $\lambda_2 - \lambda_1 = 0$, because the determinant of the matrix will be zero in that case, so that inversion is not possible. However, this is not a problem for us, because we are currently considering only the case of distinct roots, $\lambda_2 \neq \lambda_1$. Therefore, the solution to our second-order differential equation (4) with initial conditions (5) and (6), for the case when $a^2 > 4b$, is

$$y(t) = \frac{K_0\lambda_2 - K_1}{\lambda_2 - \lambda_1} e^{\lambda_1(t-t_0)} + \frac{-\lambda_1 K_0 + K_1}{\lambda_2 - \lambda_1} e^{\lambda_2(t-t_0)}. \qquad (8)$$

Next, we test if the initial conditions are satisfied through substitution,

$$y(t_0) = \frac{K_0\lambda_2 - K_1}{\lambda_2 - \lambda_1} + \frac{-\lambda_1 K_0 + K_1}{\lambda_2 - \lambda_1} = \frac{K_0(\lambda_2 - \lambda_1)}{\lambda_2 - \lambda_1} = K_0$$
$$\left.\frac{dy}{dt}\right|_{t=t_0} = \frac{K_0\lambda_2 - K_1}{\lambda_2 - \lambda_1}\lambda_1 + \frac{-\lambda_1 K_0 + K_1}{\lambda_2 - \lambda_1}\lambda_2 = \frac{K_1(\lambda_2 - \lambda_1)}{\lambda_2 - \lambda_1} = K_1,$$

which confirm that (5) and (6) are satisfied by (8).

Second-order, Case II (Critically Damped): When the roots of the characteristic polynomial are real and equal, the solution takes the form [1],

$$\lambda^2 + a\lambda + b = 0$$
$$a^2 - 4b = 0$$
$$\lambda_1 = \lambda_2 = -\frac{a}{2}$$
$$y(t) = (c_1 + c_2 t)e^{-at/2}.$$

This is known as the critically damped case, because for a fixed value of *b*, the time constant in the exponential is as small as possible, without introducing the oscillations present in the underdamped case. Let us modify this solution as well, by replacing the constants,

$$c_1 \to (c_1 - c_2 t_0)e^{at_0/2} \qquad \text{and} \qquad c_2 \to c_2 e^{at_0/2}.$$

Then the solution becomes

$$\begin{aligned} y(t) &= \left[(c_1 - c_2 t_0)e^{at_0/2} + c_2 e^{at_0/2} t\right] e^{-at/2} \\ &= \left[(c_1 - c_2 t_0) + c_2 t\right] e^{-a(t-t_0)/2} \\ &= \left[c_1 + c_2 (t - t_0)\right] e^{-a(t-t_0)/2} \end{aligned} \tag{9}$$

which simplifies the task of solving for $\{c_1, c_2\}$.

To confirm that (9) solves (4), we take the same approach as for Case I,

$$y(t) = \left[c_1 + c_2 (t - t_0)\right] e^{-a(t-t_0)/2}$$

$$\begin{aligned} \frac{dy}{dt} &= c_2 e^{-a(t-t_0)/2} - \frac{a}{2}\left[c_1 + c_2 (t - t_0)\right] e^{-a(t-t_0)/2} \\ &= \left[c_2 - \frac{a}{2}c_1 - \frac{a}{2}c_2 (t - t_0)\right] e^{-a(t-t_0)/2} \end{aligned}$$

$$\begin{aligned} \frac{d^2y}{dt^2} &= \left[-\frac{a}{2}c_2\right] e^{-a(t-t_0)/2} + \left[c_2 - \frac{a}{2}c_1 - \frac{a}{2}c_2 (t - t_0)\right]\left[-\frac{a}{2}\right] e^{-a(t-t_0)/2} \\ &= \left[-ac_2 + \frac{a^2}{4}c_1 + \frac{a^2}{4}c_2 (t - t_0)\right] e^{-a(t-t_0)/2} \end{aligned}$$

Using these expressions, the differential equation becomes,

$$\frac{d^2y}{dt^2} + a\frac{dy}{dt} + by(t) = \left[-ac_2 + \frac{a^2}{4}c_1 + \frac{a^2}{4}c_2\left(t-t_0\right)\right]e^{-a(t-t_0)/2}$$
$$+a\left[c_2 - \frac{a}{2}c_1 - \frac{a}{2}c_2\left(t-t_0\right)\right]e^{-a(t-t_0)/2} + b\left[c_1 + c_2\left(t-t_0\right)\right]e^{-a(t-t_0)/2}$$

$$= \left[-ac_2 + ac_2 + \frac{a^2}{4}c_1 - \frac{a^2}{2}c_1 + bc_1\right]e^{-a(t-t_0)/2}$$
$$+\left[\frac{a^2}{4} - \frac{a^2}{2} + b\right]c_2\left(t-t_0\right)e^{-a(t-t_0)/2}$$

$$= \left[-\frac{a^2}{4} + b\right]c_1e^{-a(t-t_0)/2} + \left[-\frac{a^2}{4} + b\right]c_2\left(t-t_0\right)e^{-a(t-t_0)/2}$$

However, we know that $a^2 = 4b$ in this case, which further implies that
$$-\frac{a^2}{4} + b = 0,$$
so that all of the terms on the right-hand side are zero. This confirms that (9) solves (4) when $a^2 = 4b$.

Applying the initial conditions, we have
$$y\left(t_0\right) = \left[c_1 + c_2\left(0\right)\right]e^{-a(0)/2} = c_1$$
$$c_1 = K_0$$
$$\frac{dy}{dt} = \left[c_2 - \frac{a}{2}c_1 - \frac{a}{2}c_2\left(t-t_0\right)\right]e^{-a(t-t_0)/2}$$
$$\left.\frac{dy}{dt}\right|_{t=t_0} = \left[c_2 - \frac{a}{2}c_1 - \left(0\right)\right]e^0 = K_1$$
$$c_2 = \frac{a}{2}c_1 + K_1 = \frac{a}{2}K_0 + K_1.$$
Therefore, the solution to (4), subject to the initial conditions (5) and (6), for the case when $a^2 = 4b$, is
$$y\left(t\right) = \left[K_0 + \left(\frac{a}{2}K_0 + K_1\right)\left(t-t_0\right)\right]e^{-a(t-t_0)/2}. \qquad (10)$$

Next, we confirm that (10) satisfies (5) and (6). Evaluating directly,
$$y\left(t_0\right) = \left[K_0 + \left(\frac{a}{2}K_0 + K_1\right)\left(0\right)\right]\left(1\right) = K_0,$$
so that (5) is satisfied, and

$$\frac{dy}{dt} = \left[\frac{a}{2}K_0 + K_1\right]e^{-a(t-t_0)/2} + \left[K_0 + \left(\frac{a}{2}K_0 + K_1\right)(t-t_0)\right]\left[-\frac{a}{2}\right]e^{-a(t-t_0)/2}$$

$$\left.\frac{dy}{dt}\right|_{t=t_0} = \left[\frac{a}{2}K_0 + K_1\right]e^0 + \left[K_0 + (0)\right]\left[-\frac{a}{2}\right]e^0 ,$$

$$= K_1$$

So that (6) is also satisfied.

<u>Second-order, Case III (Underdamped)</u>: When $a^2 < 4b$ the roots of the characteristic polynomial are complex conjugates,

$$\lambda^2 + a\lambda + b = 0$$

$$a^2 - 4b < 0$$

$$\lambda_1, \lambda_2 = -\frac{a}{2} \pm j\sqrt{b - \frac{a^2}{4}}.$$

Let

$$\sigma = -\frac{a}{2} \qquad \text{and} \qquad \omega = \sqrt{b - \frac{a^2}{4}},$$

so that

$$\lambda_1, \lambda_2 = \sigma \pm j\omega.$$

In this case, the solution to (4) takes the form

$$y(t) = e^{\sigma t}\left(A\cos\omega t + B\sin\omega t\right).$$

This is known as the underdamped case, because while the time constant of the exponential is small, the sinusoidal component introduces oscillations in the output that are undesirable for some applications. To simplify the process of finding the unspecified constants, we replace them as follows

$$A \to \left[A\cos(\omega t_0) - B\sin(\omega t_0)\right]e^{-\sigma t_0} \qquad \text{and} \qquad B \to \left[A\sin(\omega t_0) + B\cos(\omega t_0)\right]e^{-\sigma t_0}.$$

Then the solution becomes

$$y(t) = e^{\sigma t}\left\{\left[A\cos(\omega t_0) - B\sin(\omega t_0)\right]e^{-\sigma t_0}\cos(\omega t) + \left[A\sin(\omega t_0) + B\cos(\omega t_0)\right]e^{-\sigma t_0}\sin(\omega t)\right\}$$

$$= e^{\sigma(t-t_0)}\left\{\left[A\cos(\omega t_0) - B\sin(\omega t_0)\right]\cos(\omega t) + \left[A\sin(\omega t_0) + B\cos(\omega t_0)\right]\sin(\omega t)\right\}$$

$$= e^{\sigma(t-t_0)}\begin{Bmatrix} A\cos(\omega t)\cos(\omega t_0) + A\sin(\omega t)\sin(\omega t_0) \\ +B\sin(\omega t)\cos(\omega t_0) - B\cos(\omega t)\sin(\omega t_0) \end{Bmatrix}$$

$$= e^{\sigma(t-t_0)}\left\{A\cos\left(\omega\left[t - t_0\right]\right) + B\sin\left(\omega\left[t - t_0\right]\right)\right\}. \tag{11}$$

The last step above uses the familiar trigonometric identities for $\sin(x-y)$ and $\cos(x-y)$. Note that the time shift $t-t_0$ appears in place of t in all functional arguments, which illustrates the benefit of this approach.

Now we must verify that (11) solves (4) for any choice of *A* and *B*. The volume of notation becomes quite cumbersome if we attack this directly, so instead let $y_1(t)$ be a solution consisting only of the cosine term in (11), with $A=1$,

$$y_1(t) = e^{\sigma(t-t_0)}\cos(\omega[t-t_0])$$

$$\frac{dy_1}{dt} = \sigma e^{\sigma(t-t_0)}\cos(\omega[t-t_0]) - \omega e^{\sigma(t-t_0)}\sin(\omega[t-t_0])$$

$$\frac{d^2y_1}{dt^2} = \sigma^2 e^{\sigma(t-t_0)}\cos(\omega[t-t_0]) - 2\sigma\omega e^{\sigma(t-t_0)}\sin(\omega[t-t_0]) - \omega^2 e^{\sigma(t-t_0)}\cos(\omega[t-t_0]).$$

We can show that $y_1(t)$ is a solution of (4),

$$\begin{aligned}\frac{d^2y_1}{dt^2} + a\frac{dy_1}{dt} + by_1(t) &= \left[\sigma^2 - \omega^2 + a\sigma + b\right]e^{\sigma(t-t_0)}\cos(\omega[t-t_0]) \\ &\quad + [-2\sigma\omega - a\omega]e^{\sigma(t-t_0)}\sin(\omega[t-t_0]) \\ &= \left[\frac{a^2}{4} - b + \frac{a^2}{4} - \frac{a^2}{2} + b\right]e^{\sigma(t-t_0)}\cos(\omega[t-t_0]) \\ &\quad + [a\omega - a\omega]e^{\sigma(t-t_0)}\sin(\omega[t-t_0]) \\ &= 0.\end{aligned}$$

Next, let $y_2(t)$ be a solution consisting only of the sine term in (11), with $B=1$,

$$y_2(t) = e^{\sigma(t-t_0)}\sin(\omega[t-t_0])$$

$$\frac{dy_2}{dt} = \sigma e^{\sigma(t-t_0)}\sin(\omega[t-t_0]) + \omega e^{\sigma(t-t_0)}\cos(\omega[t-t_0])$$

$$\frac{d^2y_2}{dt^2} = \sigma^2 e^{\sigma(t-t_0)}\sin(\omega[t-t_0]) + 2\sigma\omega e^{\sigma(t-t_0)}\cos(\omega[t-t_0]) - \omega^2 e^{\sigma(t-t_0)}\sin(\omega[t-t_0])$$

We can also show that $y_2(t)$ is a solution of (4),

$$\begin{aligned}\frac{d^2y_2}{dt^2}+a\frac{dy_2}{dt}+by_2(t) &= [2\sigma\omega+a\omega]e^{\sigma(t-t_0)}\cos(\omega[t-t_0]) \\ &\quad+\left[\sigma^2-\omega^2+a\sigma+b\right]e^{\sigma(t-t_0)}\sin(\omega[t-t_0]) \\ &= [a\omega-a\omega]e^{\sigma(t-t_0)}\cos(\omega[t-t_0]) \\ &\quad+\left[\frac{a^2}{4}-b+\frac{a^2}{4}-\frac{a^2}{2}+b\right]e^{\sigma(t-t_0)}\sin(\omega[t-t_0]) \\ &= 0.\end{aligned}$$

Now we must demonstrate that any linear combination of $y_1(t)$ and $y_2(t)$ is also a solution of (4). This is in fact a general result that is true for any linear, homogeneous, ordinary differential equation, even without constant coefficients [1, Sec. 2.1], but we will show explicitly that it is true for our case. Therefore, let

$$y(t)=Ay_1(t)+By_2(t)$$

$$\frac{dy}{dt}=A\frac{dy_1}{dt}+B\frac{dy_2}{dt}$$

$$\frac{d^2y}{dt^2}=A\frac{d^2y_1}{dt^2}+B\frac{d^2y_2}{dt^2}$$

$$\begin{aligned}\frac{d^2y}{dt^2}+a\frac{dy}{dt}+by(t) &= A\left[\frac{d^2y_1}{dt^2}+a\frac{dy_1}{dt}+by_1(t)\right]+B\left[\frac{d^2y_2}{dt^2}+a\frac{dy_2}{dt}+by_2(t)\right] \\ &= A[0]+B[0].\end{aligned}$$

Since the right side is zero, this shows that (11) solves (4), for any constants *A* and *B*.

Next, we must determine *A* and *B* to satisfy the initial conditions (5) and (6).

$$\begin{aligned}y(t_0)=e^0\{A\cos(0)+B\sin(0)\}&=K_0 \\ A&=K_0\end{aligned}$$

$$\frac{dy}{dt} = \sigma e^{\sigma(t-t_0)}\left\{A\cos\left(\omega[t-t_0]\right) + B\sin\left(\omega[t-t_0]\right)\right\}$$
$$+ e^{\sigma(t-t_0)}\left\{-\omega A\sin\left(\omega[t-t_0]\right) + \omega B\cos\left(\omega[t-t_0]\right)\right\}$$

$$= \{\sigma A + \omega B\} e^{\sigma(t-t_0)}\cos\left(\omega[t-t_0]\right) + \{\sigma B - \omega A\} e^{\sigma(t-t_0)}\sin\left(\omega[t-t_0]\right)$$

$$\left.\frac{dy}{dt}\right|_{t=t_0} = \{\sigma A + \omega B\} e^{0}\cos(0) + \{\sigma B - \omega A\} e^{0}\sin(0)$$

$$= \sigma A + \omega B = K_1$$

$$B = \frac{K_1 - \sigma A}{\omega}$$
$$= \frac{K_1 - \sigma K_0}{\omega}$$

Therefore, the solution to (4), subject to (5) and (6), when $a^2 < 4b$, can be written

$$y(t) = e^{\sigma(t-t_0)}\left\{K_0\cos\left(\omega[t-t_0]\right) + \frac{K_1 - \sigma K_0}{\omega}\sin\left(\omega[t-t_0]\right)\right\}. \tag{12}$$

To summarize:

- The first-order initial value problem is described by (1) and (2), and the solution to this problem is presented in (3).
- The second-order problem is described by (4)-(6), with the solutions presented in (8), (10) and (12), and one determines which of these solutions to implement based on whether the value of $a^2 - 4b$ is positive, zero, or negative.

Programming Concepts: The goal of this assignment is to develop a program that can solve first-order and second-order differential equations, of the type described above. The solution will be a function of time, and the program must write the mathematical expression for the solution to a file, and generate a plot of the solution and save the plot to an image file. The signal class from Assignments 10 and 11 will be very useful for this purpose, and an object from the signal class will serve as the solution to each differential equation specified by the user.

Separate classes will be developed for first-order and second-order differential equations, because the solution techniques are different for different orders. However, there are several pieces of information that are common to both orders.

- First-order and second-order differential equations both require a coefficient b that scales the signal that is the solution to the equation, e.g. $by(t)$ in (1) and (4).

- Both orders require an initial time t_0 and an initial value for the solution, e.g. $y(t_0) = K_0$ in (2) and (5).
- Both orders will have a solution that is an object from the signal class.
- Both orders will require a label for the solution, e.g. "`w`" if the solution is $w(t)$.
- Both orders will need to write information about the differential equation and solution to text files and image files. There will be a filename template used for this purpose, in a manner similar to what was done for the signal class in assignments 10 and 11.

What tasks will needed to be completed with differential equations? The primary goal is to solve the differential equation, which can be performed by creating an object from the signal class, and filling the time vector with time values, and the signal vector with signal values, that correspond to the signal that solves the differential equation. This task must be completed in different ways for first-order and second-order equations, and some other tasks also have this property.

- There will be functions to generate a string representing the differential equation, and a string representing the initial conditions. These functions must be different for first-order and second-order equations.
- There will be a function that writes information about the differential equation and initial conditions to a text file, which will be different for different orders.

Some tasks will be common to both first-order and second-order equations, and can be implemented in the same way.

- There must be mutator and accessor functions for all of the member variables that are common to both classes.
- Once the differential equation has been solved, and the solution signal has been filled with the proper values, the program must write signal samples to a file in csv format, and the MathGL libraries must be used to create a plot of the solution. These tasks can be completed in the same way for either order.
- There will be a function that writes the mathematical expression for the solution signal to a text file. Once the solution has been defined and filled with values, this task can be completed in the same way for either order.

Separate classes will be developed to represent first-order and second-order differential equations, and because there is so much information and functionality that is common to both, it makes sense to define a base class from which the first-order and second-order classes are derived. This base class will have all of the member variables and member functions that are common to both orders.

In the derived classes, the same functionality is required of each order, but it must be implemented in different ways for different orders. This is a good opportunity to implement *polymorphism* in the program, by having member functions with the same name for both orders, but different function definitions for each order. You will write these as *virtual member functions*, so that *dynamic binding* is used to determine which function to call at runtime, as determined by the object making the call.

The base class for this assignment will represent a differential equation that can be of either first-order or second-order. However, it will not be necessary to define an instance of this class, for what does it mean to have a differential equation for which the order is not specified? The program only needs to define instances of the derived classes, so the base class will be an *abstract base class*. This means that we must define at least one member function of the base class to be a *pure virtual member function*, which is defined using "` = 0`" notation, and must be overridden in the derived classes. We will choose to define all of the member function that have different implementations for first-order and second-order in this way, although it is not necessary to do so.

To summarize, there will be three different categories for the member functions of the base class and derived classes:

- Member functions that require identical implementations for first-order and second-order equations will be defined only in the base class.
- Member functions that are required only for second-order differential equations will be defined only in the derived class for second order. (There are no member functions that are required only for first order.)
- Member functions that are required for both classes with different implementations will be given the same name for both derived classes, and will be defined as virtual member functions. For each of these, there will also be a pure virtual member function in the base class.

The signal class will figure prominently in the completion of this assignment, because it will be used to confirm and visualize the solutions. In several steps of the procedure outlined below, you will work with the signal class, to confirm that it has all the functionality required to create, describe and display the signals that are solutions to first-order and second-order differential equations. Several new member functions will be developed, and many existing member functions will be modified to customize their use with differential equations.

You will also develop the base class for a differential equation, and the first-order derived class. At the completion of these steps, you will be able to define first-order differential equations, and document and visualize the solutions. Finally, you will develop the second-order derived class and give it the additional functionality needed to solve these equations.

Instructions: Develop software according to the following specifications and submit whatever portion you have completed to the class repository before midnight on the due date.

1. Inspect the class specification file for the signal class. Make sure that this file is surrounded by an include guard (as all class specification files must be). This class will need to access functions from the libraries for `iostream`, `fstream`, `vector` and `cmath`, and the header file must have pre-processor directives to include all of them. This class will also need access to the names for `string`, `vector` and `cout`, so you may want to include `using` statements for each of these names.

   If your file includes the MathGL libraries, **delete** this pre-processor directive.

Make sure that the signal class has all of the required member variables.

- A variable of type `int` to store the number of samples.
- Two variables of type `double` to store the sampling frequency and initial time.
- Three variables from the `string` class to store the signal label, output filename template, and the mathematical expression for the signal.
- Two variables from the `vector` class of type `double` to store the time samples and signal samples.

Make sure that you have `const` accessor functions for the number of samples, sampling frequency, and initial time, and the three member variables from the `string` class. Make sure that you have `const` accessor functions for the time vector and signal vector, and for single elements from those vectors.

Make sure that you have mutator functions for the number of samples, sampling frequency, initial time, signal label and mathematical expression. Make sure that you have member functions to fill the time vector with values, and to fill the signal vector with constant or sinusoidal values. Make sure that you have member functions to write time and signal samples to a file in csv format, and to round a signal to integer values.

Make sure that you have the member function `textnum` that was provided to you for In-Class Assignment 11. This function does not modify the class instance, therefore it should have the key word `const` included at the end of the prototype. This was not required for ICA #11.

If the signal class has a member function that uses MathGL to draw a plot of the signal and save it to an image file, **delete** the prototype for this function. You will add graphics capabilities to the class in later steps.

2. Inspect the class implementation file for the signal class. Make sure that you have mutator functions for the signal label and the mathematical expression. The mutator functions for the mathematical expression sets the corresponding member variable to the value of the `string` parameter. The mutator function for the signal label also must perform this task, and then set the file name template to “`diff_eqn_soln_`” with the signal label appended to the end. This is different from the filename that was used in ICA #11.

   Make sure that you have a member function that fills the time vector with linearly-spaced values, beginning at the initial time, and consistent with the sampling rate.

   When any of the scalar member variables (number of samples, sampling frequency, or initial time) is changed, the time and signal vectors need to be updated. The mutator functions for these variables must be edited as follows (this functionality was not required for assignments 10 and 11.)

   - The mutator function for the sample rate must set the sample rate, and then call the member function to fill the time vector with values.

- The mutator function for the initial time must set the initial time, and then call the member function to fill the time vector with values.
- The mutator function for the number of samples must:
    - set the number of samples,
    - change the sizes of the time and signal vectors to be equal to the number of samples, using the member function `resize()` from the `vector` class, and
    - call the member function to fill the time vector with values.

Make sure that you have two constructors for the signal class: a default constructor that takes no parameters, and an alternate constructor that accepts one parameter of type `int` for the number of samples and two parameters of type `double` for the sampling rate and initial time. Both constructors must set all values using the mutator functions you have written for this purpose, so that they perform the additional functionality described above whenever an object from the signal class is defined.

To be specific, the default constructor must perform the following tasks:

- Call the mutator function for the number of samples, and set it to 101. (This will also set the size of the time and signal vectors, and fill the time vector with values. However the time values will be junk data, because the sampling rate and initial time have not been set yet.)
- Call the mutator function for the sampling rate, and set it to 100 samples per sec. (This will also fill the time vector with values. However the time values will be junk data, because the initial time has not been set yet.)
- Call the mutator function for the initial time, and set it to 0 sec. This will also fill the time vector with linearly-spaced values, beginning at the initial time, and consistent with the sampling rate.
- Call the mutator function for the signal label, and set it to “`x`”. This will also set the filename template to “`diff_eqn_soln_x`”.
- Call the mutator function for the mathematical expression, and set it to “`0`”.

The constructor that accepts parameters must be identical to the default constructor, except that it passes the value of the input parameters to the corresponding mutator functions for the member variables.

Make sure that your file includes a modified version of the member function `textnum` that was provided to you for In-Class Assignment 11. Shown below is the modified function that returns 3 digits for numbers less than 100, and handles both positive and negative numbers accurately (this was corrected from the Fall 2019 version of ICA #11). Please use this version of the function in your signal class.

```
string Signal::textNum(double x) const
{
     if (x >= 100)
          // large numbers truncated as int -> string
          return std::to_string(int(x));
     else
```

```
        {
             // small numbers will get 3 digits
             string x_exp = std::to_string(x);
             // return 4 characters, or 5 if x<0
             return x_exp.substr(0, 4 + (x<0));
        }
    }
```

Inspect the member function that writes information about a signal to a text file. This function must declare an `ofstream` object, and use it to open a file with the filename template and a "`.txt`" extension. The format for this file is the same as in Assignment 11, as reproduced below.

```
Time-Domain Signal Samples
N = 101
fs = 100
t0 = 0
Signal label: x
Mathematical expression
x(t) = 0
Time and signal samples in .csv format
t, x(t)
-------
0, 0
0.01, 0
0.02, 0
0.03, 0
.
.
.
0.99, 0
1, 0
```

If the signal class has a member function that uses MathGL to draw a plot of the signal and save it to an image file, **delete** the definition for this function, and make sure it is saved in a location where you can find it. You will add graphics capabilities to the class in later steps.

Write a `main` function that declares an object from the signal class without parameters, so that the default constructor is called, and then calls the member function to write signal information to a file. Run the program and make sure that the output file is as shown above.

Modify your `main` function so that the signal object is defined with parameters for the number of samples, sampling frequency, and initial time. Vary the values for these parameters, and test your program each time, to ensure that the text file contains the correct information.

Modify your `main` function so that, before writing signal information to the file, the signal label is set to something other than "`x`". Test your program, and ensure that the correct signal label is used in the output file.

3. Write a new member function for the signal class that fills the signal vector with values for the exponential signal,

$$s(t) = Ae^{-(t-t_0)/\tau},$$

   where τ is the time constant and A is the value of the signal at $t = t_0$, both of which are passed as parameters to the function. This member function must also set the mathematical expression accordingly.

   Modify your `main` function so that a signal is created with 51 samples taken at 200 samples per second, and with an initial time of $t_0 = +10$ ms. Set the signal label to "`y`", and create an exponential signal with a time constant of $\tau = 0.25$ sec and an initial value of $y(t_0) = +5$. Call the member function to write the signal to the output file.
   Run your program and inspect the output file. The correct output file is summarized below, and the signal values you obtain should be approximately equal to what is shown.

```
Time-Domain Signal Samples
N = 51
fs = 200
t0 = 0.01
Signal label: y
Mathematical expression
y(t) = 5.00 exp( -(t - 0.01) / 0.25 )
Time and signal samples in .csv format
t, y(t)
-------
0.01, 5
0.015, 4.90099
0.02, 4.80395
0.025, 4.70882
   .
   .
   .
0.255, 1.87656
0.26, 1.8394
```

4. In this step, you will begin to develop the classes for differential equations, with a base class for a differential equation of unspecified order, and a derived class for a first-order equation. At this step, we will focus on the parameters of the equation and initial condition, and on making the two classes interact properly, and all of the functionality will be in the base class.

   Create a class specification file and a class implementation file for a differential equation of unspecified order. The name of the class must be `Lhdeccic`. The private member variables are the parameter b in equations (1) and (4) and the initial time t_0 and initial value K_0 in equations (2) and (5), all of type `double`.

Write mutator functions and `const` accessor functions for the member variables.

Write a default constructor that accepts no parameters, and sets $t_0 = 0$ and $b = K_0 = 1$, by calling each of the mutator functions.

Write a second constructor that accepts three parameters, and sets the member variables to the values of those parameters by calling the mutator functions.

Create a class specification file and a class implementation file for a first-order differential equation. This must be a derived class, where the differential equation of unspecified order is the base class. The name of the class must be `Folhdeccic`. At this step, there are no member variables for the derived first-order class, other than those that are inherited from the base class.

Write a default constructor for the first-order class that calls the default constructor for the base class (and does nothing else). Write a second constructor for the first-order class that accepts three parameters, and passes their values to the constructor for the base class that accepts parameters. The body of both first-order constructors will be empty at this step.

Write a `main` function that defines an object from the first-order class, without parameters, so that the default constructor is called. Use the accessor functions to return the values of b, t_0 and K_0, and print them to standard output. Define a second object from the first-order class, using parameters, so that the alternate constructor is called. Use the accessor functions to return the values of b, t_0 and K_0, and print them to standard output.

Run your program, and confirm that the parameter values for each object are reported correctly.

Note that, at this step, we have not defined any virtual functions yet, and the base class is not an abstract base class. We could define objects from the base class at this point without generating a compile error, but there is no reason to do so.

5. In this step, you will give the differential equations classes the ability to write information about a differential equation to a text file.

   Add a private member variable to the base class. This variable is an object from the `string` class to store a label for the signal that is the solution to the differential equation. For example, if the differential equation were

$$\frac{dy}{dt} + by(t) = 0,$$

   then the signal label would be “`y`”.

Write a mutator function and a `const` accessor function for the new member variable in the base class.

Modify both constructors for the base class, so that the signal label is set to "`y`", by calling the corresponding mutator function.

Copy the member function `textNum` from the signal class, and make it part of the base class, including the prototype and definition in their respective files. This must be a `const` member function in both classes.

Add a `virtual` member function to the first-order derived class that takes no parameters, and returns an object from the `string` class. This function also will not modify the class instance, and should include the key word `const` in the prototype and definition. This function returns a text representation for the differential equation. For example, if $b = 10$ and the solution signal label is "`w`", then this function should return the string

```
w'(t) + 10.00 w(t) = 0
```

Add a `virtual` member function to the first-order derived class that takes a reference to an `ofstream` object as the only parameter, and returns nothing. The parameter must reference a file that has already been opened for writing. This function will not modify the class instance, and should include the key word `const` in the prototype and definition. This function writes four lines to the output file, the first three of which are:

```
---------------------------------
First-Order Differential Equation
---------------------------------
```

The fourth line written to the file contains only the text representation of the differential equation (as returned by the member function described above) and a newline character.

Go back to the base class and declare two virtual functions that have the same names and function prototypes as the two member functions described above. Make both of these pure virtual functions by using "` = 0`" notation in the class specification file. These functions will not appear in the class implementation file. This will also make the base class an abstract base class.

Write a `main` function that performs the following tasks:

- Define an ofstream object, and use it to open a text file named
  `ECE 0301 - Differential Equation Reports.txt`
- Define an object from the first-order class, without parameters, so that the default constructor is called. Call the member function to write information about the differential equation to a text file.
- Define an object from the first-order class, with $b = 10$, $t_0 = -2.5$ sec and $K_0 = 1$. Set the solution signal label to "`x`". Call the member function to write information about the differential equation to a text file.
- Close the text file.

Run your program and inspect the output file. It should contain the following text:

```
---------------------------------
First-Order Differential Equation
---------------------------------
y'(t) + 1.00 y(t) = 0

---------------------------------
First-Order Differential Equation
---------------------------------
x'(t) + 10.0 x(t) = 0
```

6. In this step, you will give the differential equations classes the ability to report information about the initial condition, and write this information in a text file.

   Add a `virtual` member function to the first-order class that takes no parameters, and returns an object from the `string` class. This function will not modify the class instance, and should include the key word `const` in the prototype and definition. This function returns a text representation for the initial condition. For example, if $t_0 = -2.5$, $K_0 = 1$, and the solution signal label is "`w`", then this function should return the string

   ```
   w(-2.500) = 1.000
   ```

   In the base class, declare a `virtual` member function that has the same name and function prototype as the member function described above. Include the `const` keyword in the prorotype. Make this a pure virtual function by using "` = 0`" notation.

   Return to the first-order derived class, and modify the function that writes information about a differential equation to a text file. After completing the tasks described in step 9, have the function write additional lines to the file to report the initial condition, following the format shown below.

   You should not have to modify your `main` function for this step. After the program is run, the output file should contain the following text:

```
---------------------------------
First-Order Differential Equation
---------------------------------
y'(t) + 1.00 y(t) = 0

Initial Condition
-----------------
y(0.00) = 1.00

---------------------------------
First-Order Differential Equation
---------------------------------
x'(t) + 10.0 x(t) = 0
```

```
Initial Condition
-----------------
x(-2.50) = 1.00
```

7. In this step, you will give the differential equations classes the ability to define an object from the signal class, fill it with samples of the solution signal, and write information about the solution to a text file. The first-order differential equation is described by (1), with the initial condition described by (2). The solution is given in (3), from which we note that the solution will always be an exponential function.

   In order to complete this task, we must choose the time range over which to compute the solution, the number of samples to compute, and the sampling rate to use. For this assignment, you are required to compute the solution for $t_0 \le t \le t_0 + 5\tau$, where τ is the time constant of the exponential signal. You must also compute 100 samples of the solution signal per time constant, for a total of 501 samples. Use this information to determine the number of samples, initial time, and sampling frequency when defining an object from the signal class that will store the solution.

   In the base class, add a member variable that is an object from the signal class, which will be used to store the solution to the differential equation.

   In the base class, modify the mutator function for the solution signal label. Add a statement to this file that uses the solution signal to set its label member variable to the label used for the differential equation. This will ensure that, whenever the signal label for a differential equation is changed, the signal label for the solution will be updated.

   In the base class, add a member function that takes no parameters, and returns a pointer to an object from the signal class. This function returns the member variable that is a pointer to the solution signal. This function is needed so that the derived classes will be able to access and modify the solution signal. You **must not** include the keyword `const` in the prototype or the definition for this function.

   In the first-order class, add a `virtual` member function that fills the solution signal with values, according to equation (3), by performing the following tasks:
   - Define a pointer to an object from the signal class, and set it to point to the solution signal, by calling the member function of the base class that returns the solution pointer.
   - Use the solution pointer to set the number of samples, sampling rate, and initial time, consistent with the requirements described above.
   - Use the solution pointer to call the member function from the signal class that fills a signal with exponential values. Pass values to the parameters of this function so that the signal is filled with values according to equation (3).

   In the first-order class, add one statement to both constructors that calls the member function to fill the solution signal with values. Since the bodies of both constructors were empty until now, this is the only statement that will appear in the bodies of the constructors in the first-order class. By including this statement, as each differential equation is defined, the solution signal will be immediately filled with values.

Return to the base class, and declare a `virtual` member function that has the same name and function prototype as the member function in the first-order class that fills the solution signal with exponential values. Make this a pure virtual function by using "` = 0`" notation.

In the base class, make the mutator functions for b, t_0 and K_0 `virtual`. In the first-order class, add `virtual` mutator functions for b, t_0 and K_0. Each of these functions must call the corresponding mutator function from the base class, and then call the member function from the first-order class to fill the solution signal with values. By doing so, any time one of the parameters of a first-order differential equation is changed, the solution will be immediately updated. It is not possible to do this by modifying the mutator functions in the base class, because the member function that fills the solution signal with exponential signals is part of the first-order derived class.

In the base class, add a member function that takes no arguments, and returns an object from the `string` class. This function returns a text representation for an equation that describes the solution to the differential equation. Since it will not modify the class instance, this function should have the key word `const` included at the end of the prototype and header. The string returned by this function can be assembled by concatenating three strings together:

- the member variable from the base class that stores the solution signal label,
- the string constant "`(t) = `", and
- the member variable from the signal class that stores the mathematical expression for the signal.

In the base class, add a `virtual` member function that writes information about the solution of a differential equation to a text file. This function takes a reference to an `ofstream` object as the only parameter, and returns nothing. The parameter must reference a file that has already been opened for writing. This function writes lines to the output file, following the format shown below, so that the solution will also be included in the file. Since it will not modify the class instance, this function should have the key word `const` included at the end of the prototype and header.

Modify your `main` function so that, for each differential equation, after calling the member function to write the differential equation and initial conditions to the text file, the member function that writes the solution to the file is also called. Run your program, and inspect the output file. If your program is working correctly, the output file should contain:

```
--------------------------------
First-Order Differential Equation
--------------------------------
y'(t) + 1.00 y(t) = 0

Initial Condition
```

```
   ----------------
   y(0.00) = 1.00

   Solution
   --------
   y(t) = 1.00 exp( -t  )

   --------------------------------
   First-Order Differential Equation
   --------------------------------
   x'(t) + 10.0 x(t) = 0

   Initial Condition
   -----------------
   x(-2.50) = 1.00

   Solution
   --------
   x(t) = 1.00 exp( -(t + 2.50) / 0.10 )
```

When you are certain your program is correct, save you main function in a file named

```
   ece0301_inclass13_step07.cpp
```

Submit this file, and all of the following files, to the class repository.

```
   Signal.h
   Signal.cpp
   Lhdeccic.h
   Lhdeccic.cpp
   Folhdeccic.h
   Folhdeccic.cpp
```

8. In the base class, examine the function that writes information about the solution to the differential equation to the output file. Add a statement to the end of this function that uses the solution signal to call the member function from the signal class that writes signal information to a file. By including this statement, each time a differential equation writes is used to write information to a text file, both the text file for the differential equation and the text file for the solution signal will be updated.

   In the first-order derived class, examine the function that fills the solution signal with exponential values. Add a statement to the end of this function that rounds the solution signal values to the nearest integer. By including this statement, the solution signal will always be rounded to integer values, which will make it easier to compare the solution signal values with the values in the example output files posted on the course website.

   Modify your `main` function so that, after opening the output file, an object from the first-order derived class is defined that corresponds to the initial value problem

$$\frac{dx}{dt} + 2x(t) = 0 \qquad \text{subject to} \qquad x(-1.25) = 120\,.$$

Next, call the member function to write information about the differential equation to the output file, and then call the member function that writes information about the solution to the differential equation to the output file. This will, in turn, call the member function for the solution signal object, so that information about this signal is written to a separate text file.

Run your program, and examine the text file `ECE 0301 - Differential Equation Reports.txt`. The contents should be as follows:

```
---------------------------------
First-Order Differential Equation
---------------------------------
x'(t) + 2.00 x(t) = 0

Initial Condition
-----------------
x(-1.25) = 120

Solution
--------
x(t) = 120 exp( -(t + 1.25) / 0.50 )
```

In addition, your program should have generated the file `diff_eqn_soln_x.txt`. The correct contents for this file are summarized below.

```
Time-Domain Signal Samples
N = 501
fs = 200
t0 = -1.25
Signal label: x
Mathematical expression
x(t) = 120 exp( -(t + 1.25) / 0.50 )
Time and signal samples in .csv format
t, x(t)
-------
-1.25, 120
-1.245, 119
-1.24, 118
.
.
.
1.24, 1
1.245, 1
1.25, 1
```

When you are certain your program is correct, save you main function in a file named

```
ece0301_inclass13_step08.cpp
```

Submit this file, and all of the following files, to the class repository.

```
Signal.h
Signal.cpp
```

```
Lhdeccic.h
Lhdeccic.cpp
Folhdeccic.h
Folhdeccic.cpp
```

9. Make sure you save the files from step 8 saved in a convenient location for later use. Do not modify them for the remaining steps, make new copies of your files for this purpose.

   Modify the signal class by adding the capability to plot a signal using Math GL.
   - Add a pre-processor directive to `Signal.h` to include the MathGL libraries.
   - Add the prototype for the plotting function to `Signal.h`.
   - Add the definition for the plotting function to `Signal.cpp`.

   In the base class, examine the member function that writes information about the solution to the differential equation to the output file. Add a statement to the end of this function that uses the solution signal to call the member function from the signal class that draws a plot of the signal to an image file using MathGL.

   You should not have to modify your main function for this step. When you run your program, you should find that it creates the same text files shown in step 8. In addition, it should create `.png` and `.eps` files containing a plot of the solution signal, with all values rounded to integers. The image should appear similar to Figure 1.

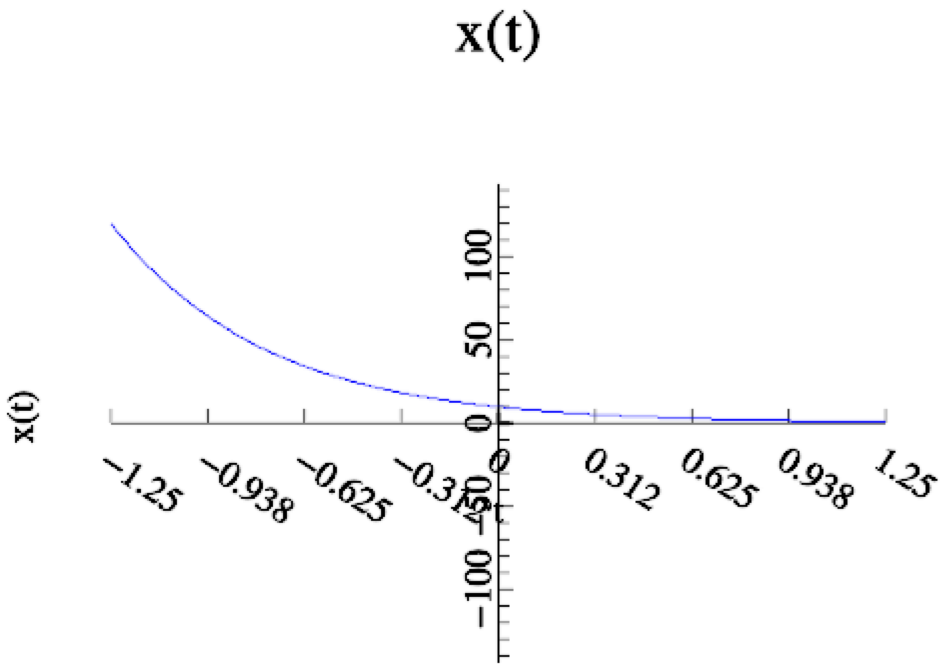


Figure 1: Plot of solution for first-order differential equation.

10. Over the remaining steps, we will develop the class for a second-order differential equation that will represent the equation and initial conditions described by equation (4) – (6). We will start by focusing on the overdamped case, where $a^2 > 4b$, and the solution is given by equation (8), i.e. the sum of two exponential signals. Therefore, in this this step, you will return to the signal class to ensure that it is capable of defining and plotting such signals.

    Make sure that you have overloaded the = operator for the signal class. Make sure that your operator function accepts a `const` reference object from the signal class as a parameter. Make sure that the return type is a `const` object from the signal class. Make sure that, if there is no self-assignment, this operator function copies the proper member variables (and no others) from the parameter object to the calling object:
    - Number of samples, sampling rate, and initial time.
    - Time vector and signal vector.
    - Mathematical expression.

    Make sure that this function returns the contents of the `this` pointer.

    Make sure that you have overloaded the + operator for the signal class. Make sure that your operator function accepts a `const` reference object from the signal class as a parameter. Make sure that the return type is an object from the signal class (not `const`). This function must check if the calling object and parameter object have the same number of samples, sampling rate, and initial time. If these are all true, then the operator function must declare a new signal object, give it the same time vector as the calling object, and set the signal vector to the sum of the signals from the calling object and the parameter object. It must also form the mathematical expression for the sum of the two signals, as in ICA #11.

    Write a main function to create a signal that is the sum of two exponentials, by performing these tasks:
    - Create a signal that has 201 samples, taken at 20 samples per sec, with an initial time of 0.5 sec. Fill this signal with an exponential signal, with a time constant of 2 sec, and an initial value of 0.4. Set the signal label to "`w`". This object will represent the time-domain signal
    $$w(t) = 0.4e^{-(t-0.5)/2} \qquad \text{for} \qquad 0.5 \le t \le 10.5$$
    - Create a new signal that has the same number of samples, sampling rate and initial time as the first. Fill this signal with an exponential signal, with a time constant of 1 sec, and an initial value of -0.4. Set the signal label to "`v`". This object will represent the time-domain signal
    $$v(t) = -0.4e^{-(t-0.5)} \qquad \text{for} \qquad 0.5 \le t \le 10.5$$
    - Use the overloaded = and + operators to form the sum of the two signals, and store it back in the first signal, i.e. to implement
    $$w(t) = w(t) + v(t)$$
    - Call the member functions to write $w(t)$ to a text file and plot it using MGL.

    Run your program and examine the image file it produces. If your program is correct, the plot should appear similar to Figure 2.

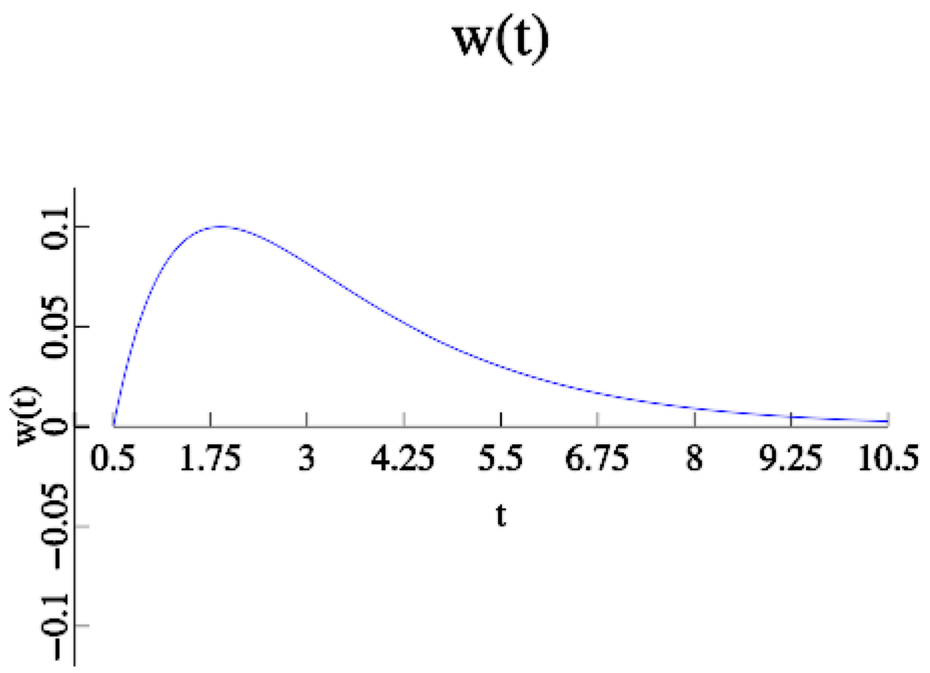


Figure 2: Plot of sum of two exponential signals.

Examine the output text file produced by your program. The mathematical expression for the signal should be

```
  w(t) = 0.40 exp( -(t - 0.50) / 2.00 ) - 0.40 exp( -(t - 0.50) )
```

and the time and signal values should be consistent with Figure 2.

11. In this step, you will begin to develop the class for a second-order differential equation. For now, objects from this class will be given the ability to write the differential equation and initial conditions to a text file.

    Create a class specification file and a class implementation file for a second-order differential equation. This must be a derived class, where the differential equation of unspecified order is the base class. This class will inherit all of the member variables and functions from the base class, and must also have two private member variables of type double to store the coefficient *a*, and the initial value K_1 for the derivative of the solution.

    Write mutator functions and `const` accessor functions for the member variables of the second-order derived class.

    Write two constructors for the second-order class. The default constructor must call the default constructor for the base class, and then set the parameters to $a = 3$, $b = 2$, $t_0 = K_0 = 0$, and $K_1 = 1$, and set the signal label to "`y`", by calling the mutator functions for these member variables.

The alternate constructor must accept five parameters of type `double`, representing desired values for a, b, t_0, K_0 and K_1. This function must call the alternate constructor for the base class, passing along the values for b, t_0 and K_0, and then set the values for a and K_1, and set the signal label to “`y`”, by calling the mutator functions for these member variables.

Write a `virtual` member function that accepts no parameters, and returns an object from the `string` class. This function will not modify the class instance, and should include the key word `const` in the prototype and definition. This function returns a text representation for the second-order differential equation, and must have the same name as the function in the first-order class that performs the same task. For example, if $a = 2$, $b = -101$ and the solution signal label is “`z`”, then this function should return the string

```
z''(t) + 2.00 z'(t) - 101 z(t) = 0
```

Write a `virtual` member function that accepts no parameters, and returns an object from the `string` class. This function will not modify the class instance, and should include the key word `const` in the prototype and definition. This function returns a text representation for the initial conditions, and must have the same name as the function in the first-order class that performs the same task. For example, if $t_0 = 0$, $K_0 = 1$, $K_1 = 10$, and the solution signal label is “`z`”, then this function should return the string

```
z(0.00) = 1.00, z'(0.00) = 10.0
```

Write a `virtual` member function that takes a reference to an `ofstream` object as the only parameter, and returns nothing. The parameter must reference a file that has already been opened for writing. This function writes information about a second-order differential equation and the initial conditions to the text file, and it must have the same name as the member function in the first-order class that performs the same task. This function will not modify the class instance, and should include the key word `const` in the prototype and definition. The text format used by this function is very similar to that used for the first-order class, as shown in the example output below.

Write a `virtual` member function that takes no parameters, and returns nothing. In later steps, this function will fill the solution signal with values, but for this step the body of this function must be empty. The reason why you must do this is because there is a pure virtual function in the base class with the same name, and this function must be overridden by a function in the second-order derived class.

Modify your `main` function by deleting the statements that define objects from the first-order differential equation class, and call their member functions. Insert statements in their place that perform the following tasks:

- Define an ofstream object, and use it to open a text file named
  ```
  ECE 0301 - Differential Equation Reports.txt
  ```
- Define an object from the second-order differential equations class, without parameters, so that the default constructor is called.

- Call the member function that writes the differential equation and initial conditions to a text file.
- Define an object from the second-order differential equations class, with parameters, so that the alternate constructor is called, and it defines the initial value problem

$$\frac{d^2z}{dt^2}+2\frac{dz}{dt}-101z(t)=0 \qquad \text{subject to} \qquad z(0)=1 \quad \text{and} \quad z'(0)=10$$

- Use the differential equation object to set the label to "`z`".
- Call the member function that writes the differential equation and initial conditions to a text file.

Run your program, and inspect the output file. If your program is working correctly, the output file should contain:

```
---------------------------------
Second-Order Differential Equation
---------------------------------
y''(t) + 3.00 y'(t) + 2.00 y(t) = 0

Initial Condition
----------------
y(0.00) = 0.00, y'(0.00) = 1.00

---------------------------------
Second-Order Differential Equation
---------------------------------
z''(t) + 2.00 z'(t) - 101 z(t) = 0

Initial Condition
----------------
z(0.00) = 1.00, z'(0.00) = 10.0
```

12. In this step, you will give the second-order differential equation class the ability to fill the solution signal with values, write the mathematical expression for the solution to a text file, and draw a plot of the solution using MathGL. We will begin with the overdamped case, because the default constructor generates a differential equation with $a=3$ and $b=2$, so that $a^2>4b$.

    In the second-order class, edit the `virtual` member function that fills the solution signal with values. The body of this function should be empty, and you must add statements that fill the solution vector with samples of the solution defined by equation (8). The solution must be computed for $t_0 \le t \le t_0+5\tau$, where $\tau = \max\{\tau_1,\tau_2\}$ and τ_1,τ_2 are the time constants of the two exponential terms, and 100 samples of the solution signal per τ, for a total of 501 samples.
    - Define a pointer to an object from the signal class, and set it to point to the solution signal, by calling the member function of the base class that does so.

- Use the solution pointer to set the number of samples and initial time consistent with the requirements described above.
- Use the quadratic formula to compute the rates for the two exponential terms, i.e. the roots of the characteristic polynomial,
$$\lambda^2 + a\lambda + b = 0\,.$$
Since we are dealing with the overdamped case here, $a^2 > 4b$, so the two roots you obtain should be real-valued and distinct.
- Use the solution pointer to set the sample rate for the solution signal so that 100 samples are computed for each τ, where $\tau = \max\{\tau_1, \tau_2\}$ and τ_1, τ_2 are the time constants of the two exponential terms.
- Compute the coefficients $c_1 = \dfrac{K_0\lambda_2 - K_1}{\lambda_2 - \lambda_1}$ and $c_2 = \dfrac{-\lambda_1 K_0 + K_1}{\lambda_2 - \lambda_1}$.
- Use the solution pointer to call the member function from the signal class that fills a signal with exponential values. Pass values to the parameters of this function so that the signal is filled with the first exponential term in equation (8).
- Define a new object from the signal class, with the same number of samples, sample rate and initial time as the solution signal. Fill this signal with exponential values, corresponding to the second term in equation (8).
- Use the overloaded + and = operators to add the two signals, and store the result back in the solution signal.

Edit the two constructors for the second-order class so that the last statement is a call to the member function that fills the solution signal with values.

Modify your `main` function so that only one second-order differential equation is defined, using the default constructor. Call the member functions for this object to:

- write the differential equation and initial conditions to the text file, and
- write information about the solution to the differential equation to a text file. This will also call the member functions from the signal class to write samples of the solution to a file in .csv format, and draw a plot of the signal to image files using MathGL.

Run your program and inspect the output files. The output text file for the default second-order differential equation should contain:

```
----------------------------------
Second-Order Differential Equation
----------------------------------
y''(t) + 3.00 y'(t) + 2.00 y(t) = 0

Initial Condition
-----------------
y(0.00) = 0.00, y'(0.00) = 1.00

Solution
--------
```

```
   y(t) = 1.00 exp( -t ) - 1.00 exp( -t / 0.50 )
```

The image generated by the solution to the default second-order differential equation should appear as shown in Figure 3.

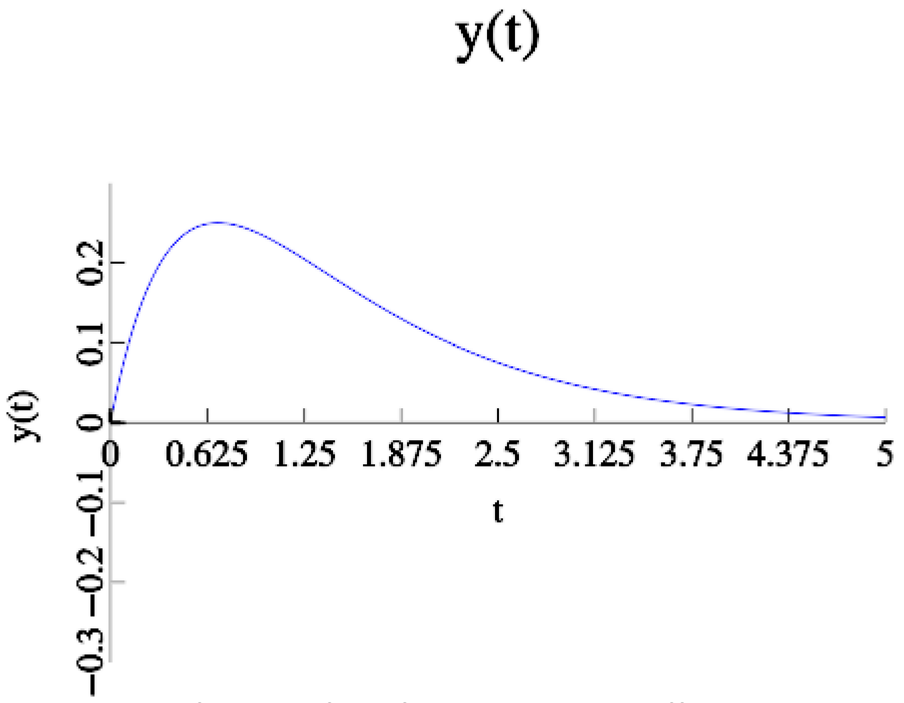


Figure 3: Plot of solution for default second-order differential equation.

The text file produced by your program that contains samples of the solution signal should include the mathematical expression

```
   y(t) = 1.00 exp( -t ) - 1.00 exp( -t / 0.50 )
```

and the signal values should be consistent with Figure 3.

13. In this step, you will remove the ability of your program to plot the solution using MathGL, which is necessary for automated testing purposes. You should save the functions necessary for plotting signals in a convenient location for later use.

    Edit the class specification file for the signal class. Delete the pre-processor directive to include the MathGL libraries from this file (and any other files where you have included it.) Delete the prototype for the function that plots a signal using MathGL.

    Edit the class implementation file for the signal class. Delete the definition for the function that plots a signal using MathGL.

    Edit the class implementation file for the base class that defines a differential equation of unspecified order. Examine the function that reports information about the solution to the differential equation. Delete the (last) statement in this function that calls the member function from the signal class that was just deleted (to plot the signal).

Edit the class implementation file for the derived class that defines a differential equation of second order. Examine the function that fills the solution signal with values. Add a statement to the end of this function that uses the solution pointer to call the member function from the signal class that rounds the signal values to the nearest integers.

Edit your `main` function so that it defines the initial value problem

$$\frac{d^2y}{dt^2}+3\frac{dy}{dt}+2y(t)=0 \qquad \text{subject to} \qquad y(0)=0 \quad \text{and} \quad y'(0)=250$$

The only change from step 12 is the value of K_1. Run your program and examine the output files. The output file generated by the differential equation object should contain:

```
----------------------------------
Second-Order Differential Equation
----------------------------------
y''(t) + 3.00 y'(t) + 2.00 y(t) = 0

Initial Condition
-----------------
y(0.00) = 0.00, y'(0.00) = 250

Solution
--------
y(t) = 250 exp( -t ) - 250. exp( -t / 0.50 )
```

The output file generated by the solution signal object is summarized below:

```
Time-Domain Signal Samples
N = 501
fs = 100
t0 = 0
Signal label: y
Mathematical expression
y(t) = 250 exp( -t ) - 250. exp( -t / 0.50 )
Time and signal samples in .csv format
t, y(t)
-------
0, 0
0.01, 2
0.02, 5
.
.
.
4.98, 2
4.99, 2
5, 2
```

The output files for this step are also available on the course website. When you are certain your program is correct, save you main function in a file named

```
ece0301_inclass13_step13.cpp
```

Submit this file, and all of the following files, to the class repository.

```
Signal.h
Signal.cpp
Lhdeccic.h
Lhdeccic.cpp
Folhdeccic.h
Folhdeccic.cpp
Solhdeccic.h
Solhdeccic.cpp
```

14. Our next goal is to expand the functionality of the second-order derived class to handle the critically damped case, where $a^2 = 4b$, and the solution is given by equation (10). Over the next two steps, you will modify the signal class so that it is capable of producing the signal described by equation (10).

    Return to the signal class. Write a new member function that fills the signal vector with values for a signal that varies linearly with time according to

    $$s(t) = m(t - t_0) + b\,,$$

    where *m* is the slope and *b* is the value of the signal at $t = t_0$, both of which are passed as parameters to the function. The mathematical expression produced by this function must be consistent with the equation shown above.

    Modify your `main` function so that a signal is created with 501 samples taken at 100 samples per second, and with an initial time of -2.5 sec. Set the signal label to "`s`", and fill the signal with values according to

    $$s(t) = 750(t + 2.5) - 150$$

    Call the member functions to write the signal to a file.

    Run your program and inspect the output file, which is summarized below.

```
Time-Domain Signal Samples
N = 501
fs = 100
t0 = -2.5
Signal label: s
Mathematical expression
s(t) = 750 (t + 2.50) - 150
Time and signal samples in .csv format
t, s(t)
-------
-2.5, -150
-2.49, -142.5
```

```
-2.48, -135
.
.
.
2.48, 3585
2.49, 3592.5
2.5, 3600
```

15. Make sure that you have overloaded the `*` operator for the signal class. Make sure that your operator function accepts a `const` reference object from the signal class as a parameter. Make sure that the return type is an object from the signal class (not `const`). This function must check if the calling object and parameter object have the same number of samples, sampling rate, and initial time. If these are all true, then the operator function must declare a new signal object, give it the same time vector as the calling object, and set the signal vector to the product of the signals from the calling object and the parameter object. It must also form the mathematical expression for the sum of the two signals, as in ICA #11.

    Write a main function to create a signal that is in the form of equation (10), by performing these tasks:

    - Create a signal that has 501 samples, taken at 100 samples per sec, with an initial time of -2.5 sec. Fill this signal with a linear signal, with a slope of 750, and an initial value of -150. Set the signal label to "`s`". This object will represent the time-domain signal
    $$s(t) = 750(t+2.5) - 150 \qquad \text{for} \qquad -2.5 \le t \le 2.5$$
    - Create a new signal that has the same number of samples, sampling rate and initial time as the first. Fill this signal with an exponential signal, with a time constant of 0.75 sec, and an initial value of 1. Set the signal label to "`v`". This object will represent the time-domain signal
    $$v(t) = e^{-(t+2.5)/0.75} \qquad \text{for} \qquad -2.5 \le t \le 2.5$$
    - Use the overloaded = and * operators to form the product of the two signals, and store it back in the first signal, i.e. to implement
    $$s(t) = s(t) \cdot v(t)$$
    - Call the member function to write $s(t)$ to a text file.

Run your program and inspect the output file produced by the signal. The correct output file is summarized below.

```
Time-Domain Signal Samples
N = 501
fs = 100
t0 = -2.5
Signal label: s
Mathematical expression
s(t) = (750 (t + 2.50) - 150)(1.00 exp( -(t + 2.50) / 0.75 ))
Time and signal samples in .csv format
t, s(t)
-------
-2.5, -150
-2.49, -140.613
-2.48, -131.448
.
.
.
2.48, 4.68569
2.49, 4.6333
2.5, 4.58148
```

16. Edit the class implementation file for the derived class that defines a differential equation of second order. Examine the function that fills the solution signal with values. Modify this function so that it performs the following tasks:
    - Define a pointer to an object from the signal class, and initialize it to point to the solution signal, by calling the member function from the base class that returns the solution signal pointer.
    - Recall that we will compute each solution signals over five time constants, with 100 samples per time constant, for a total of 501 samples. Set the number of samples for the solution signal accordingly.
    - Set the initial time for the solution signal to the initial time for the differential equation.
    - If $a^2 > 4b$, your function must implement the tasks from step 12, so that the solution signal is filled with the sum of two exponential signals, as required for the overdamped case. This includes setting the sampling rate for the solution signal so that there 100 samples per τ, where $\tau = \max\{\tau_1, \tau_2\}$ and τ_1, τ_2 are the time constants of the two exponential terms.
    - If $a^2 = 4b$, your function must fill the solution signal as described by equation (10):
      - Compute $\lambda = -a/2$.
      - Set the sample rate for the solution signal so that that there are 100 samples per τ, where $\tau = 1/|\lambda|$.
      - Compute $c_1 = K_0$ and $c_2 = K_1 + \frac{K_o a}{2}$.
      - Create a signal $c_2(t - t_0) + c_1$.

  - Create a second signal $e^{\lambda t}$
  - Set the solution signal equal to the product of the two signals just created.
- Call the member function from the signal class to round the solution signal to the nearest integers.

Modify your `main` function so that it performs the following tasks:

- Define an ofstream object, and use it to open a text file named
  `ECE 0301 - Differential Equation Reports.txt`
- Define an object from the second-order differential equations class, with parameters, so that the alternate constructor is called, and it defines the initial value problem

$$\frac{d^2s}{dt^2}+4\frac{ds}{dt}+4s(t)=0 \qquad \text{subject to} \qquad s(0)=-50 \quad \text{and} \quad s'(0)=500$$

- Use the differential equation object to set the label to “`s`”.
- Call the member function that writes the differential equation and initial conditions to a text file.
- Call the member function that writes information about the solution to the differential equation to a text file. This will also call the member function from the signal class to write samples of the solution to a file in .csv format.

Run your program and inspect the output files. The text file generated by the differential equation object should contain the following:

```
----------------------------------
Second-Order Differential Equation
----------------------------------
s''(t) + 4.00 s'(t) + 4.00 s(t) = 0

Initial Condition
-----------------
s(0.00) = -50.0, s'(0.00) = 500

Solution
--------
s(t) = (400 t - 50.0)(1.00 exp( -t / 0.50 ))
```

The output file generated by the solution signal object is summarized below:

```
Time-Domain Signal Samples
N = 501
fs = 200
t0 = 0
Signal label: s
Mathematical expression
s(t) = (400 t - 50.0)(1.00 exp( -t / 0.50 ))
Time and signal samples in .csv format
t, s(t)
-------
0, -50
```

```
0.005, -48
0.01, -45
.
.
.
2.49, 7
2.495, 6
2.5, 6
```

The output files for this step are also available on the course website. When you are certain your program is correct, save you main function in a file named

```
ece0301_inclass13_step16.cpp
```

Submit this file, and all of the following files, to the class repository.

```
Signal.h
Signal.cpp
Lhdeccic.h
Lhdeccic.cpp
Folhdeccic.h
Folhdeccic.cpp
Solhdeccic.h
Solhdeccic.cpp
```

17. Our next goal is to expand the functionality of the second-order derived class to handle the underdamped case, where $a^2 < 4b$, and the solution is given by equation (12). In this step, you will modify the signal class so that it is capable of producing the signal described by equation (12).

    Inspect the member function that fills the signal vector with values for a sinusoidal signal. Modify this function so that the signal created is includes a time shift by t_0, i.e.

    $$s(t) = A\cos\left(2\pi f_0\left(t - t_0\right) + \phi\right),$$

    where A is the amplitude, f_0 is the frequency in Hz, and and ϕ is the phase in radians, all of which are passed as parameters to the function. The mathematical expression produced by this function must also be altered accordingly.

    Write a main function to create a signal that is in the form of equation (12), by performing these tasks:

    - Create a signal that has 501 samples, taken at 100 samples per sec, with an initial time of 0 sec. Fill this signal with a sinusoidal signal, with an amplitude of 2, a frequency of 10 Hz, and a phase of 0. Set the signal label to “g”. This object will represent the time-domain signal

      $$g(t) = 2\cos(20\pi t) \qquad \text{for} \qquad 0 \le t \le 5$$

    - Create a new signal that has the same number of samples, sampling rate and initial time as the first. Fill this signal with a sinusoidal signal, with an amplitude of 2, a frequency of 10 Hz, and a phase of $-\pi/2$. Set the signal label to “h”. This object will represent the time-domain signal

$$h(t) = 2\sin(20\pi t) \qquad \text{for} \qquad 0 \le t \le 5$$

- Create a new signal that has the same number of samples, sampling rate and initial time as the first. Fill this signal with an exponential signal, with a time constant of 1 sec, and an initial value of 1. Set the signal label to “v”. This object will represent the time-domain signal

$$v(t) = e^{-t} \qquad \text{for} \qquad 0 \le t \le 5$$

- Use the overloaded =, + and * operators to form the product of the two signals, and store it back in the first signal, i.e. to implement

$$g(t) = \left[g(t) + h(t)\right] \cdot v(t)$$

- Call the member function to write $g(t)$ to a text file.

Run your program and inspect the output file produced by the signal. The correct output file is summarized below. (The mathematical expression is displayed on two lines here, but occupies only one line in the file.)

```
Time-Domain Signal Samples
N = 501
fs = 100
t0 = 0
Signal label: g
Mathematical expression
g(t) = (2.00 cos( 62.8 t  ) + 2.00 cos( 62.8 t - 1.57
))(1.00 exp( -t ))
Time and signal samples in .csv format
t, g(t)
-------
0, 2
0.01, 2.76581
0.02, 2.47024
.
.
4.98, -0.00882684
4.99, 0.00301126
5, 0.0134759
```

18. Edit the class implementation file for the derived class that defines a differential equation of second order. Examine the function that fills the solution signal with values. Modify this function so that it performs the following tasks:
    - Define a pointer to an object from the signal class, and initialize it to point to the solution signal, by calling the member function from the base class that returns the solution signal pointer.

- Recall that we will compute each solution signals over five time constants, with 100 samples per time constant, for a total of 501 samples. Set the number of samples for the solution signal accordingly.
- Set the initial time for the solution signal to the initial time for the differential equation.
- If $a^2 > 4b$, your function must implement the tasks from step 12, so that the solution signal is filled with the sum of two exponential signals, as required for the overdamped case.
- If $a^2 = 4b$, your function must implement the tasks from step 16, so that the solution signal is filled with the sum of two exponential signals, as required for the critically damped case.
- If $a^2 < 4b$, your function must fill the solution signal as described by equation (12):
  - Compute $\sigma = a/2$ and $\omega = \dfrac{\sqrt{4b-a^2}}{2}$.
  - Set the sample rate for the solution signal so that that there are 100 samples per τ, where $\tau = 1/\sigma$.
  - Compute $c_1 = K_0$ and $c_2 = \dfrac{K_1 - \sigma K_0}{\omega}$.
  - Create a signal $e^{-\sigma t}$.
  - Create a second signal $c_1 \cos(\omega t)$
  - Create a third signal $c_2 \sin(\omega t)$.
  - Use the operators to set the solution signal equal to $e^{-\sigma t}\left[c_1 \cos(\omega t) + c_2 \sin(\omega t)\right]$.
- Call the member function from the signal class to round the solution signal to the nearest integers.

Modify your `main` function so that it performs the following tasks:

- Define an ofstream object, and use it to open a text file named
  `ECE 0301 - Differential Equation Reports.txt`
- Define an object from the second-order differential equations class, with parameters, so that the alternate constructor is called, and it defines the initial value problem

$$\frac{d^2 g}{dt^2} + 2\frac{dg}{dt} + 101 g(t) = 0 \qquad \text{subject to} \qquad g(-5) = 100 \quad \text{and} \quad g'(-5) = -100$$

- Use the differential equation object to set the label to "`g`".
- Call the member function that writes the differential equation and initial conditions to a text file.
- Call the member function that writes information about the solution to the differential equation to a text file. This will also call the member function from the signal class to write samples of the solution to a file in .csv format.

Run your program and inspect the output files. The text file generated by the differential equation object should contain the following. (The mathematical expression is displayed on two lines here, but occupies only one line in the file.)

```
----------------------------------
Second-Order Differential Equation
----------------------------------
g''(t) + 2.00 g'(t) + 101 g(t) = 0

Initial Condition
-----------------
g(-5.00) = 100, g'(-5.00) = -100.

Solution
--------
g(t) = (1.00 exp( -(t + 5.00) ))(100 cos( 10.0 (t + 5.00)
) - 20.0 cos( 10.0 (t + 5.00) - 1.57 ))
```

The output file generated by the signal object is summarized below. (The mathematical expression is displayed on two lines here, but occupies only one line in the file.)

```
Time-Domain Signal Samples
N = 501
fs = 100
t0 = -5
Signal label: g
Mathematical expression
g(t) = (1.00 exp( -(t + 5.00) ))(100 cos( 10.0 (t + 5.00)
) - 20.0 cos( 10.0 (t + 5.00) - 1.57 ))
Time and signal samples in .csv format
t, g(t)
-------
-5, 100
-4.99, 97
-4.98, 92
.
.
.
-0.02, 1
-0.01, 1
0, 1
```

The output files for this step are also available on the course website. When you are certain your program is correct, save you main function in a file named

```
ece0301_inclass13_step18.cpp
```

Submit this file, and all of the following files, to the class repository.

```
Signal.h
Signal.cpp
```

```
Lhdeccic.h
Lhdeccic.cpp
Folhdeccic.h
Folhdeccic.cpp
Solhdeccic.h
Solhdeccic.cpp
```

19. In the final step of this assignment, you will restore the ability of objects from the differential equations classes to produce plots of the solution signals using MathGL, and write a main program that will define and solve a variety of initial value problems.

    Edit the class specification file for the signal class. Add a pre-processor directive to include the MathGL libraries. Add the prototype for the member function that draws a plot of the signal using MathGL. You should be able to use the same member function that was used in step 9.

    Edit the class implementation file for the signal class. Copy the member function that draws a plot of the signal using MathGL from the location where you saved it, and paste it into the class implementation file for the signal class.

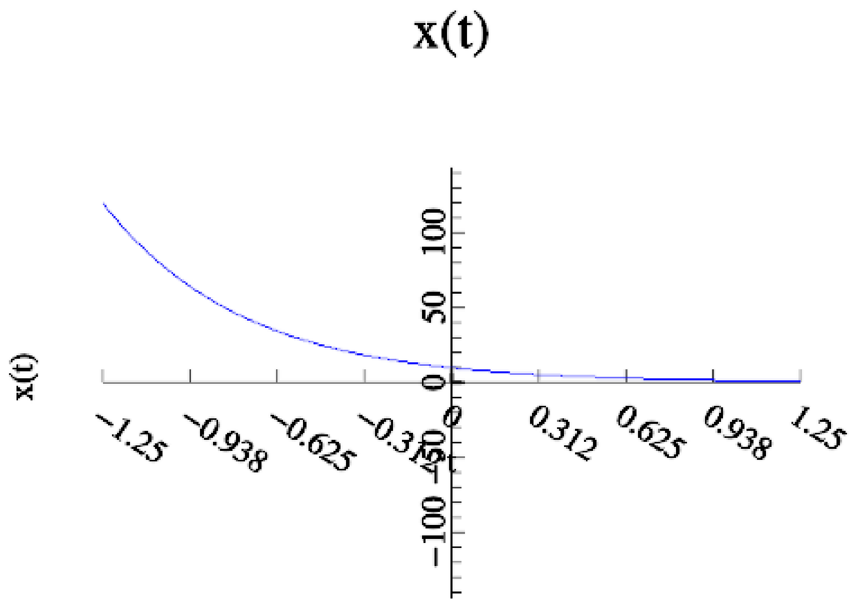
    Edit the class implementation file for the base class that defines a differential equation of unspecified order. Examine the member function that reports information about the solution to the differential equation. Add a statement to the end of this function that calls the member function from the signal class to draw a plot of the solution signal using MathGL.

    Modify your `main` function so that your program solves all of the following initial value problems. For each problem, you must:

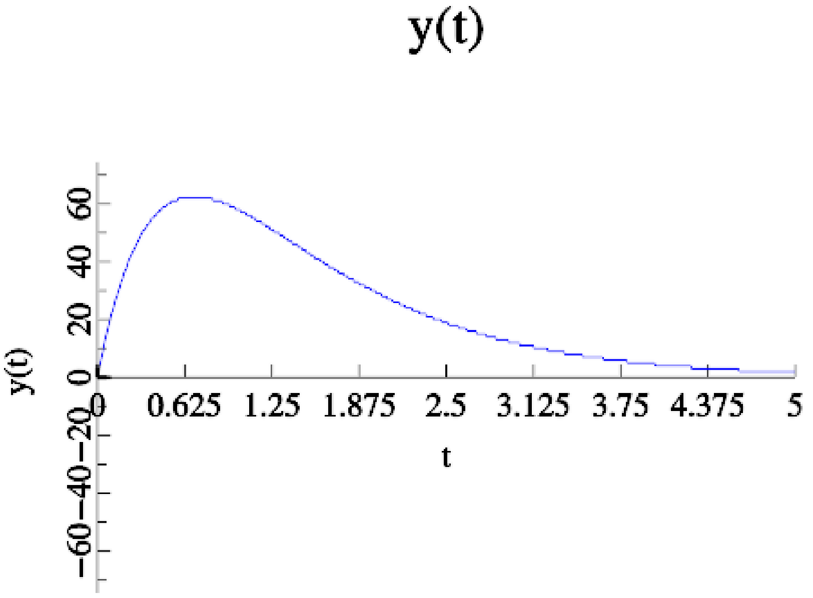
    - Define an appropriate object from the differential equations classes,
    - Call the member function that writes information about differential equation and initial conditions to a text file.
    - Call the member function that reports information about the solution to the differential equation. This will also call the member functions that write the solution signal to a text file, and plot it using MathGL.

    Confirm that each solution and plot is correct.

(a) $$\frac{dx}{dt}+2x(t)=0$$
$$x(-1.25)=120$$



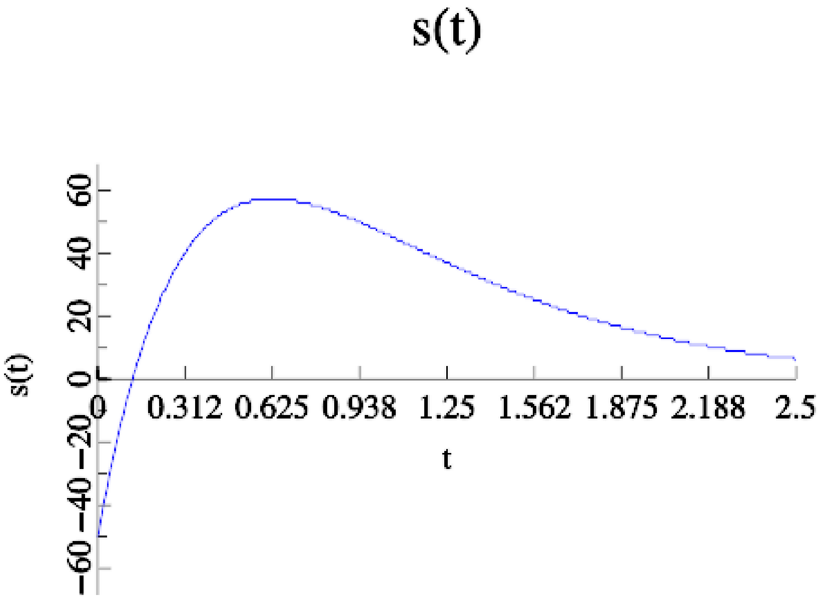
(b) $$\frac{d^2y}{dt^2}+3\frac{dy}{dt}+2y(t)=0$$
$$y(0)=0$$
$$\left.\frac{dy}{dt}\right|_{t=0}=250$$



$$\frac{d^2s}{dt^2}+4\frac{ds}{dt}+4s(t)=0$$

(c) $$s(0)=-50$$

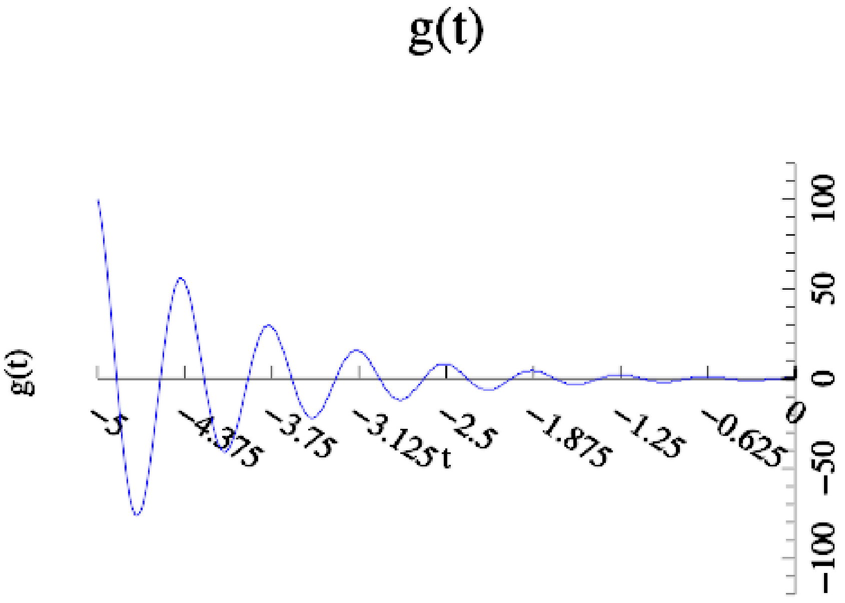
$$\left.\frac{ds}{dt}\right|_{t=0}=500$$



$$\frac{d^2g}{dt^2}+2\frac{dg}{dt}+101g(t)=0$$

(d) $$g(-5)=100$$

$$\left.\frac{dg}{dt}\right|_{t=-5}=-100$$


g(t)
g(t)
100
50
0
-50
-100
-5
-4.375
-3.75
-3.125
-2.5
-1.875
-1.25
-0.625
0
t