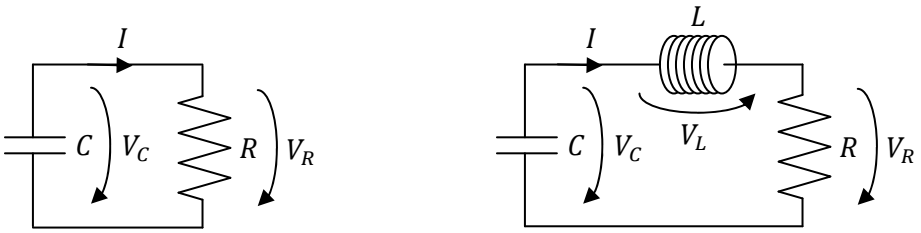
# The Rayleigh Pulse Forming Network

The ideal power supply for many high current applications is one which supplies a square voltage wave over a short, but predetermined, time. Since these applications typically require that a large amount of energy be available over a short period of time, it is usually necessary to store the energy in advance in a capacitor bank, and to trigger a sudden discharge when the event is to be powered. Since the load is usually some kind of resistance, or a use of power which can be modeled as a resistance, the sudden discharge will follow the classical exponential capacitor-resistor discharge curve. This gives a huge initial spike in current, which decreases exponentially as time passes. This exponential decay is usually undesirable. A more suitable power supply would "slow down" the delivery of the current, so that it becomes available to the load resistor over a longer period of time.

Since inductors resist changes in the flow of current, they can be used in combination with the capacitor bank to create a more desirable current waveform. Let us compare the two simplest cases: a capacitor discharging through a resistor, and the same capacitor discharging through the same resistor and an ideal inductor placed in series. The two configurations are shown in the following figure.




The voltage-current characteristics of the components are:

$$\left.\begin{array}{ll} \text{Capacitor:} & V_C = V_C(0) - \dfrac{1}{C}\displaystyle\int_{\tau=0}^{\tau=t} I\,d\tau \\ \text{Resistor:} & V_R = RI \\ \text{Inductor:} & V_L = L\dfrac{dI}{dt} \end{array}\right\} \quad (1)$$

Since both circuits are simple loops, the sum of the voltage drops around the loops must be zero. Adding up the voltage drops of the components, and taking the time-derivatives, gives:

$$\left.\begin{array}{ll} \text{On the left:} & RC\dfrac{dI}{dt} + I = 0 \\ \text{On the right:} & LC\dfrac{d^2I}{dt^2} + RC\dfrac{dI}{dt} + I = 0 \end{array}\right\} \quad (2)$$

Proposing a current waveform of the type $I(t) = Ae^{\alpha t}$ and substituting this waveform into the differential equations gives the following characteristic equations:

$$\left.\begin{array}{ll} \text{On the left:} & (RC\alpha + 1)e^{\alpha t} = 0 \\ \text{On the right:} & (LC\alpha^2 + RC\alpha + 1)e^{\alpha t} = 0 \end{array}\right\} \quad (3)$$

which have the solutions:

$$\left.\begin{array}{ll} \text{On the left:} & \alpha = -\dfrac{1}{RC} \\ \text{On the right:} & \alpha = \dfrac{-RC \pm \sqrt{R^2C^2 - 4LC}}{2LC} \end{array}\right\} \quad (4)$$

It is usual to define time-constants. For the resistor-capacitor pair, the time-constant $\tau_{RC}$ is defined as the product $RC$. For the resistor-inductor pair, the time constant $\tau_{RL}$ is defined as the quotient $L/R$. Using these time-constants, the exponential rates of change can be written as:

$$\left.\begin{array}{ll} \text{On the left:} & \alpha = -\dfrac{1}{\tau_{RC}} \\ \text{On the right:} & \alpha = \dfrac{-\tau_{RC} \pm \sqrt{\tau_{RC}^2 - 4\tau_{RC}\tau_{RL}}}{2\tau_{RC}\tau_{RL}} \end{array}\right\} \quad (5)$$

For the circuit on the left, $\alpha < 0$ means that the exponential waveform is a decay. The rate of the decay is such that the magnitude of the current decreases by a fraction $1/e = 36.8\%$ during each additional time interval of one time-constant.

For the circuit on the right, the waveform can have different forms. There are two roots for $\alpha$ so the current will be the sum of two different waveforms. If the term under the radical is positive, then the two roots will be real. The resulting waveform will be the sum of two exponential curves, one increasing and the other decreasing. If the term under the radical is negative, the two values of $\alpha$ will be complex conjugates. The waveform will be a sinusoid (arising from the imaginary part of $\alpha$) whose envelope is modified in an exponential way (resulting from the real part of $\alpha$).

In this paper, we are going to be interested in circuits in which the two time-constants are the same: $\tau_{RC} = \tau_{RL}$. Why this is desirable will be described below. In any event, if the two time-constants are the same, for which we will use the symbol $\tau$, then the roots for the rate of change $\alpha$ are as follows:

$$\begin{aligned} \text{On the right:} \quad \alpha &= \frac{-\tau \pm \sqrt{\tau^2 - 4\tau^2}}{2\tau^2} \\ &= \frac{-\tau \pm j\sqrt{3}\tau}{2\tau^2} \\ &= \frac{-1 \pm j\sqrt{3}}{2\tau} \end{aligned} \quad (6)$$

I have used the symbol $j$ for the imaginary number. Readers will recognize 1, 2 and $\sqrt{3}$ as the three sides of a 30° right triangle. This is one of the many fundamental relationships which swirl around this topic. Using the unknown constant coefficients $A$ and $B$ for the two solutions, we can write the waveform for the current as:

$$\begin{aligned} I(t) &= Ae^{\frac{(-1+j\sqrt{3})t}{2\tau}} + Be^{\frac{(-1-j\sqrt{3})t}{2\tau}} \\ &= e^{-\frac{t}{2\tau}}\left(Ae^{\frac{j\sqrt{3}t}{2\tau}} + Be^{-\frac{j\sqrt{3}t}{2\tau}}\right) \\ &= e^{-\frac{t}{2\tau}}\left(A'\sin\frac{\sqrt{3}t}{2\tau} + B'\cos\frac{\sqrt{3}t}{2\tau}\right) \end{aligned} \quad (7)$$

It is convenient to replace the sum of the two exponential terms which have purely imaginary arguments with their equivalent representation as the sum of two sinusoidal terms. Different coefficients $A'$ and $B'$ are needed.

Let us assume that the circuit on the right is initialized by charging up the capacitor to some voltage $V_C(0)$. We will set the time base of the analysis so that time $t = 0$, the starting instant, occurs when the current is zero: $I(0) = 0$. Since the current is initially zero, there will be no voltage drop over the resistor. The inductor will be exposed to the full voltage drop over the capacitor. This means that $V_L(0) = V_C(0)$ and, using the V-I characteristic of the inductor, we can say that the initial rate of change of the current will be equal to $V_C(0)/L$. These are the mathematical forms of the two initial conditions. Substituting time $t = 0$ the solution gives:

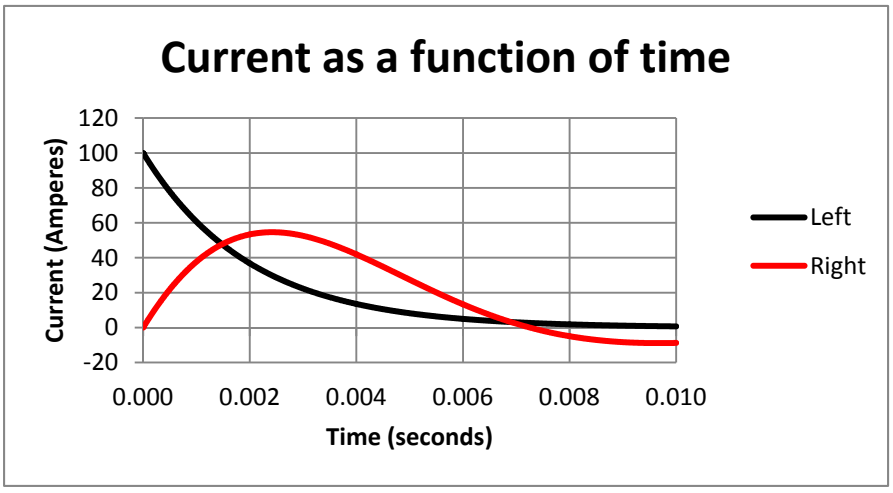
$$\begin{aligned} 1st\ IC: \quad & I(0) && = e^0(A' \sin 0 + B' \cos 0) \\ \rightarrow \quad & 0 && = B' \end{aligned} \tag{8}$$

$$\begin{aligned} 2nd\ IC: \quad & \frac{dI}{dt} && = \left\{ \begin{aligned} & -\frac{1}{2\tau} e^{-\frac{t}{2\tau}} \left( A' \sin \frac{\sqrt{3}t}{2\tau} + B' \cos \frac{\sqrt{3}t}{2\tau} \right) + \cdots \\ & \cdots + e^{-\frac{t}{2\tau}} \left( A' \frac{\sqrt{3}}{2\tau} \cos \frac{\sqrt{3}t}{2\tau} - B' \frac{\sqrt{3}}{2\tau} \sin \frac{\sqrt{3}t}{2\tau} \right) \end{aligned} \right\} \\ \rightarrow \quad & \frac{V_C(0)}{L} && = -\frac{1}{2\tau} B' + A' \frac{\sqrt{3}}{2\tau} \\ \rightarrow \quad & A' && = \frac{2\tau V_C(0)}{\sqrt{3}L} \end{aligned} \tag{9}$$

To summarize, the complete expressions for the currents in the two circuits are as follows:

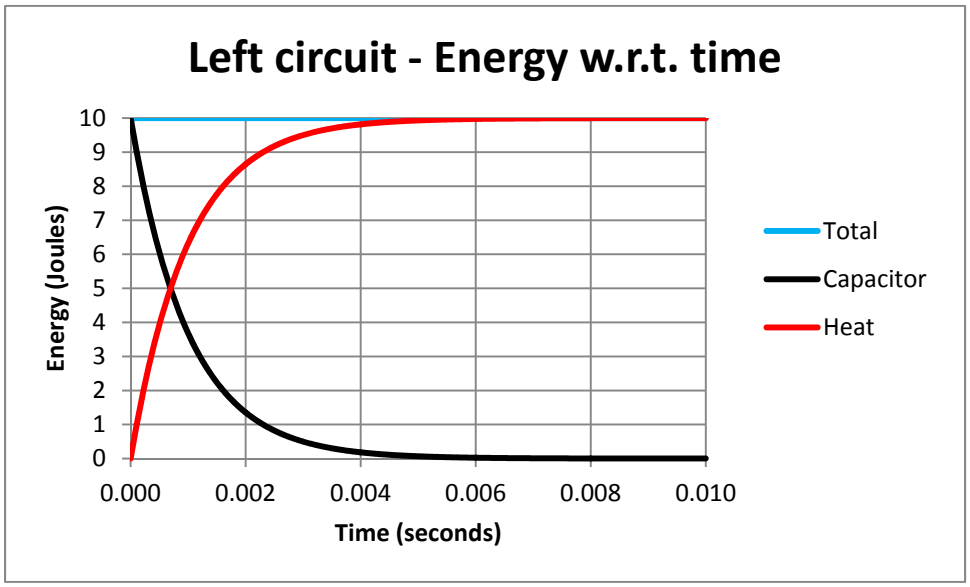
$$\left. \begin{aligned} &\text{On the left:} && I(t) = \frac{V_C(0)}{R} e^{-\frac{t}{\tau}} \\ &\text{On the right:} && I(t) = \frac{2\tau V_C(0)}{\sqrt{3}L} e^{-\frac{t}{2\tau}} \sin \frac{\sqrt{3}t}{2\tau} \end{aligned} \right\} \tag{10}$$

I have written the solution for the circuit on the left without going through the details, but have used the same time constant as for the circuit on the right. The following graph compares the two waveforms. It is necessary to choose fixed numbers in order to draw a graph. I picked $\tau = 2$ ms and $V_C(0) = 100$ V. The time constant establishes the ratio between $R$ and $L$ in such a way that $L = 0.002R$. It also fixes the ratio between $R$ and $C$ in such a way that $C = 0.002/R$. If we choose the load resistance $R$ to be one Ohm, in the expectation that the current will be in the order of $100\ \text{V}/1\ \Omega = 100$ A, then the required values for $L$ and $C$ are $L = 0.002$ H and $C = 0.002$ F.

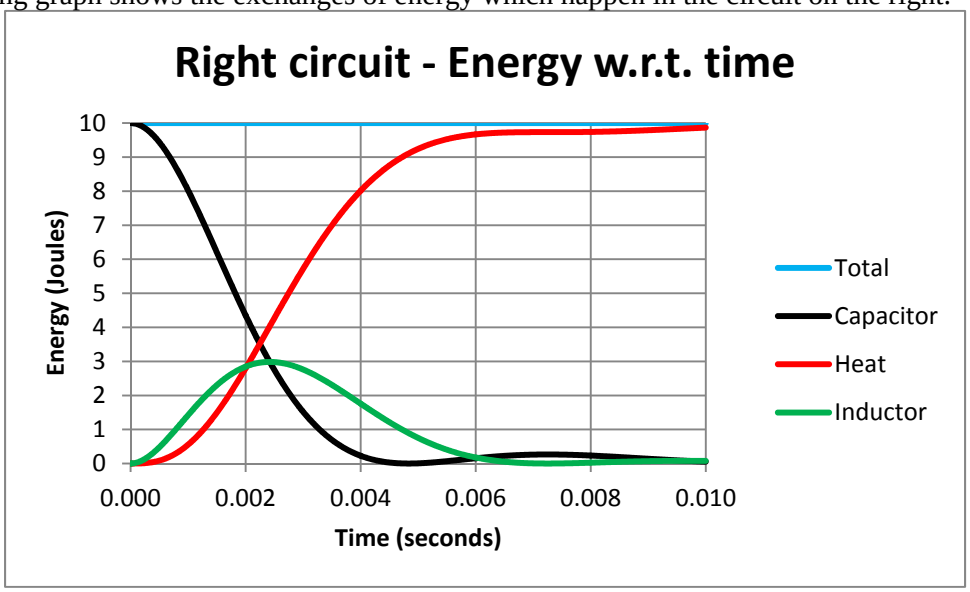


Adding the inductor to the original R-C circuit has shifted the current peak from the starting instant out to just over two milliseconds. In fact, if you think about it, the red waveform is about as close to a square wave as one can achieve using a single inductor.

It is useful to see where the energy goes as time progresses. The energy $E_C$ stored in a capacitor $C$ is given as a function of its voltage drop as $E_C = \frac{1}{2}CV_C^2$. In a similar way, the energy $E_L$ stored in the magnetic field of an inductor $L$ is given in terms of the current flowing through it as $E_L = \frac{1}{2}LI^2$. No similar statement can be made about the "energy stored" in the resistor. Resistors do not return to the circuit any of the energy they consume. Instead, they dissipate it as heat. The rate at which a resistor consumes energy is, of course, equal to its power $P_R$ and is given by $P_R = RI^2$. Calculating the cumulative energy which has been consumed by a resistor requires integrating the instantaneous power over the period of time of interest. The following graph shows the transfer of energy to the resistor in the simple R-C circuit on the left. I have labeled the energy consumed by the resistor as "heat" but, in many applications, the desired work performed by the driven apparatus will be included in this use of energy.



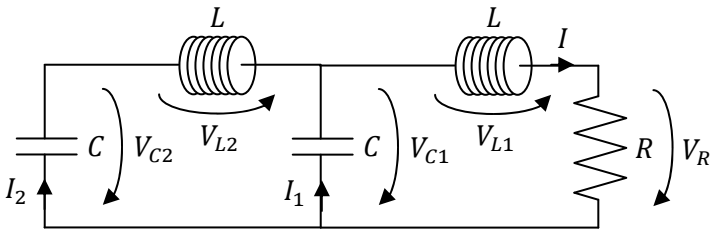

The following graph shows the exchanges of energy which happen in the circuit on the right.



For the circuit on the right, the exchange of energy can be broken down into two periods. For the first two milliseconds, the capacitor supplies energy both to the inductor and to the resistor, in approximately equal measures. After about two milliseconds, the inductor returns its energy to the circuit, so the resistor enjoys the inductor's energy plus the remaining tag-end of the capacitor's energy.

**<u>If one inductor is good, two must be better</u>**

Let us add another "stage" to this power supply. One can think about dividing the initial stock of energy between two capacitors, each of which drives an inductor. This is shown in the following diagram.




It is important to note that:

- the two coils are <u>not</u> linked by any mutual inductance,
- the two capacitors have equal values $C$ and
- the two inductors have equal values $L$.

I have distinguished between the two capacitors by labeling their voltage drops $V_{C1}$ and $V_{C2}$. Similarly, I have distinguished between the two inductors by labeling their voltage drops $V_{L1}$ and $V_{L2}$. I have identified separate currents $I_1$ and $I_2$ flowing through the separate capacitors. For convenience, I have continued to use the symbol $I$ for the current flowing through the load resistor and $V_R$ for the voltage drop over it. We now have seven circuit equations.

$$\left.\begin{array}{ll} R: & V_R = R(I_1 + I_2) \\ C_1: & V_{C1} = V_{C1}(0) - \dfrac{1}{C}\displaystyle\int_{\tau=0}^{\tau=t} I_1 d\tau \\ L_1: & V_{L1} = L\dfrac{d(I_1 + I_2)}{dt} \\ C_2: & V_{C2} = V_{C2}(0) - \dfrac{1}{C}\displaystyle\int_{\tau=0}^{\tau=t} I_2 d\tau \\ L_2: & V_{L2} = L\dfrac{dI_2}{dt} \\ \text{Loop1:} & V_{C1} = V_R + V_{L1} \\ \text{Loop2:} & V_{C2} = V_{C1} + V_{L2} \end{array}\right\} \quad (11)$$

The first step is to take the first time-derivative of all the equations. The second step is to eliminate the voltages from each of the loop equations. When these steps are done, one is left with two equations:

$$\left.\begin{array}{c} LC\dfrac{d^2(I_1 + I_2)}{dt^2} + RC\dfrac{d(I_1 + I_2)}{dt} + I_1 = 0 \\ I_1 = I_2 + LC\dfrac{d^2 I_2}{dt^2} \end{array}\right\} \quad (12)$$

The two differential equations can be unlinked, to eliminate $I_1$, by substituting $I_1$ from the second equation into the first. This gives:

$$LC\frac{d^2I_2}{dt^2} + LC\frac{d^2}{dt^2}\left(I_2 + LC\frac{d^2I_2}{dt^2}\right) + RC\frac{dI_2}{dt} + RC\frac{d}{dt}\left(I_2 + LC\frac{d^2I_2}{dt^2}\right) + \left(I_2 + LC\frac{d^2I_2}{dt^2}\right) = 0$$

$$L^2C^2\frac{d^4I_2}{dt^4} + RLC^2\frac{d^3I_2}{dt^3} + 3LC\frac{d^2I_2}{dt^2} + 2RC\frac{dI_2}{dt} + I_2 = 0 \qquad (13)$$

This is a fourth-order differential equation in current $I_2(t)$, which is the newly-added loop. An exponential form of solution, $I_2(t) = Ae^{\alpha t}$, will produce the following characteristic equation.

$$\alpha^4L^2C^2 + \alpha^3RLC^2 + 3\alpha^2LC + 2\alpha RC + 1 = 0 \quad (14)$$

A closed-form solution is not possible. Generally, a numerical solution would require that we pick fixed values for the components. That is not the case here – we do not need to specify the components just yet. In the one stage power supply above (the circuit on the right), we set the resistor-capacitor time-constant equal to the resistor-inductor time-constant. We are going to do something similar now, by taking advantage of the fact that both of the capacitances are equal and both of the inductances are equal. We are going to set:

$$\left.\begin{aligned} C &= \frac{T}{2R} \\ L &= \frac{RT}{2} \end{aligned}\right\} \quad (15)$$

where the time $T$ will be related in some way to the duration of the hoped-for square wave pulse. This choice for $C$ and $L$ causes both time-constants to be the same, as above, and causes both time-constants to be equal to $\frac{1}{2}T$. The following expansions confirm this.

$$\left.\begin{aligned} \tau_{RC} &= RC = R\frac{T}{2R} = \tfrac{1}{2}T \\ \tau_{RL} &= \frac{L}{R} = \frac{1}{R}\frac{RT}{2} = \tfrac{1}{2}T \end{aligned}\right\} \quad (16)$$

When we insert these choices for $L$ and $C$ into the characteristic equation (14), we get:

$$\tfrac{1}{16}\alpha^4T^4 + \tfrac{1}{8}\alpha^3T^3 + \tfrac{3}{4}\alpha^2T^2 + \alpha T + 1 = 0 \quad (17)$$

The beauty of Equation (17) is that we do not need to specify a duration $T$ at all. We can combine $T$ with $\alpha$ in a new variable $x = \alpha T$ to write the following polynomial.

$$\tfrac{1}{16}x^4 + \tfrac{1}{8}x^3 + \tfrac{3}{4}x^2 + x + 1 = 0 \quad (18)$$

A root-finding program available on the internet gives the four roots as:

$$\alpha T = -0.790246764 \pm j1.01368780 \quad \text{and} \quad -0.209753235 \pm j3.10498364 \quad (19)$$

The current $I_2(t)$ will be a linear combination of four terms, each based on one of these four roots. We can write the solution in sinusoidal terms using four unknown constants $G$, $H$, $P$ and $Q$, as follows:

$$I_2(t) = \begin{bmatrix} e^{-\frac{0.790246764t}{T}}\left(G\sin\frac{1.01368780t}{T} + H\cos\frac{1.01368780t}{T}\right) + \cdots \\ \cdots + e^{-\frac{0.209753235t}{T}}\left(P\sin\frac{3.10498364t}{T} + Q\cos\frac{3.10498364t}{T}\right) \end{bmatrix} \quad (20)$$

We need to circle back now and look at the other current, $I_1(t)$, in the loop containing the load resistor. The second differential equation in Equation (12) gives $I_1(t)$ directly in terms of $I_2(t)$. Using the first solution / term in Equation (20) as an example, we get:

$$\begin{aligned} I_1 &= I_2 + \tfrac{1}{4}T^2\frac{d^2I_2}{dt^2} \\ &= \begin{Bmatrix} Ge^{-\frac{0.790246764t}{T}}\sin\frac{1.01368780t}{T} + \cdots \\ \cdots + \frac{1}{4}T^2\frac{d}{dt}\begin{bmatrix} -G\frac{0.790246764}{T}e^{-\frac{0.790246764t}{T}}\sin\frac{1.01368780t}{T} + \cdots \\ \cdots + G\frac{1.01368780}{T}e^{-\frac{0.790246764t}{T}}\cos\frac{1.01368780t}{T} \end{bmatrix} \end{Bmatrix} \\ &= \begin{bmatrix} Ge^{-\frac{0.790246764t}{T}}\sin\frac{1.01368780t}{T} + \cdots \\ \cdots + \frac{1}{4}T^2G\frac{0.790246764^2}{T^2}e^{-\frac{0.790246764t}{T}}\sin\frac{1.01368780t}{T} + \cdots \\ \cdots - \frac{1}{4}T^2G\frac{0.790246764}{T}\frac{1.01368780}{T}e^{-\frac{0.790246764t}{T}}\cos\frac{1.01368780t}{T} + \cdots \\ \cdots - \frac{1}{4}T^2G\frac{1.01368780}{T}\frac{0.790246764}{T}e^{-\frac{0.790246764t}{T}}\cos\frac{1.01368780t}{T} + \cdots \\ \cdots - \frac{1}{4}T^2G\frac{1.01368780^2}{T^2}e^{-\frac{0.790246764t}{T}}\sin\frac{1.01368780t}{T} \end{bmatrix} \\ &= \begin{bmatrix} G\left(1 + \frac{1}{4}0.790246764^2 - \frac{1}{4}1.01368780^2\right)e^{-\frac{0.790246764t}{T}}\sin\frac{1.01368780t}{T} + \cdots \\ \cdots - \frac{1}{2}G(0.790246764 \times 1.01368780)e^{-\frac{0.790246764t}{T}}\cos\frac{1.01368780t}{T} \end{bmatrix} \\ &= Ge^{-\frac{0.790246764t}{T}}\begin{bmatrix} \left(1 + \frac{1}{4}0.790246764^2 - \frac{1}{4}1.01368780^2\right)\sin\frac{1.01368780t}{T} + \cdots \\ \cdots - \frac{1}{2}(0.790246764 \times 1.01368780)\cos\frac{1.01368780t}{T} \end{bmatrix} \end{aligned} \quad (21)$$

Each of the four terms will follow the same pattern. What is noteworthy is that the exponential decay of each term in $I_1(t)$ is the same as the decay of the term in $I_2(t)$ from which it is derived. Furthermore, the constant coefficient of each term in $I_1(t)$ is directly proportional to the constant coefficient of the term in $I_2(t)$ from which it is derived. This means that the overall solution, for both currents, involves only the four unknown constants. The four initial conditions will enable us to solve for them.

I have been putting it off, but now is a good time to write down the complete expansion for $I_1(t)$.

$$I_1(t) = \left\{ \begin{array}{l} Ge^{-\frac{0.790246764t}{T}} \left[ \begin{array}{c} \left(1 + \frac{1}{4}0.790246764^2 - \frac{1}{4}1.01368780^2\right) \sin\frac{1.01368780t}{T} + \cdots \\ \cdots - \frac{1}{2}(0.790246764 \times 1.01368780) \cos\frac{1.01368780t}{T} \end{array} \right] + \cdots \\ \cdots + He^{-\frac{0.790246764t}{T}} \left[ \begin{array}{c} \left(1 + \frac{1}{4}0.790246764^2 - \frac{1}{4}1.01368780^2\right) \cos\frac{1.01368780t}{T} + \cdots \\ \cdots + \frac{1}{2}(0.790246764 \times 1.01368780) \sin\frac{1.01368780t}{T} \end{array} \right] + \cdots \\ \cdots + Pe^{-\frac{0.209753235t}{T}} \left[ \begin{array}{c} \left(1 + \frac{1}{4}0.209753235^2 - \frac{1}{4}3.10498364^2\right) \sin\frac{3.10498364t}{T} + \cdots \\ \cdots - \frac{1}{2}(0.209753235 \times 3.10498364) \cos\frac{3.10498364t}{T} \end{array} \right] + \cdots \\ \cdots + Qe^{-\frac{0.209753235t}{T}} \left[ \begin{array}{c} \left(1 + \frac{1}{4}0.209753235^2 - \frac{1}{4}3.10498364^2\right) \cos\frac{3.10498364t}{T} + \cdots \\ \cdots + \frac{1}{2}(0.209753235 \times 3.10498364) \sin\frac{3.10498364t}{T} \end{array} \right] \end{array} \right\}$$

It is also noteworthy that the duration of the pulse length $T$ does not appear anywhere in the coefficients. When we compute the sums in the various brackets, $I_1(t)$ can be re-written as follows.

$$I_1(t) = \left\{ \begin{array}{l} Ge^{-\frac{0.790246764t}{T}} \left[ \begin{array}{c} 0.899231748 \sin\frac{1.01368780t}{T} + \cdots \\ \cdots - 0.400531752 \cos\frac{1.01368780t}{T} \end{array} \right] + \cdots \\ \cdots + He^{-\frac{0.790246764t}{T}} \left[ \begin{array}{c} 0.899231748 \cos\frac{1.01368780t}{T} + \cdots \\ \cdots + 0.400531752 \sin\frac{1.01368780t}{T} \end{array} \right] + \cdots \\ \cdots + Pe^{-\frac{0.209753235t}{T}} \left[ \begin{array}{c} -1.399231746 \sin\frac{3.10498364t}{T} + \cdots \\ \cdots - 0.325640182 \cos\frac{3.10498364t}{T} \end{array} \right] + \cdots \\ \cdots + Qe^{-\frac{0.209753235t}{T}} \left[ \begin{array}{c} -1.399231746 \cos\frac{3.10498364t}{T} + \cdots \\ \cdots + 0.325640182 \sin\frac{3.10498364t}{T} \end{array} \right] \end{array} \right\} \quad (22)$$

Now, let us begin to apply some initial conditions. If we start the circuit from a standing start, then both currents will be zero at time $t = 0$. By inspection, setting Equation (20) for $I_2(t)$ equal to zero requires that $H + Q = 0$. Also by inspection, setting Equation (22) for $I_1(t)$ equal to zero requires that:

$$H + Q = 0 \qquad (I.C.\#1)$$
$$-0.400531752G + 0.899231748H - 0.325640182P - 1.399231746Q = 0 \qquad (I.C.\#2)$$

Next, observe that inductor $L_2$ is a bridge between the two capacitors. Since the two capacitors will initially be charged up to the same voltage, there will be no voltage drop over $L_2$ at time $t = 0$. Since the voltage drop over $L_2$ is proportional to the derivative of the current $I_2$ flowing through it, we can write the third initial condition as follows:

$$\frac{dI_2}{dt} = \begin{bmatrix} -G\frac{0.790246764}{T}e^{-\frac{0.790246764t}{T}}\sin\frac{1.01368780t}{T} + \cdots \\ \cdots + G\frac{1.01368780}{T}e^{-\frac{0.790246764t}{T}}\cos\frac{1.01368780t}{T} + \cdots \\ \cdots - H\frac{0.790246764}{T}e^{-\frac{0.790246764t}{T}}\cos\frac{1.01368780t}{T} + \cdots \\ \cdots - H\frac{1.01368780}{T}e^{-\frac{0.790246764t}{T}}\sin\frac{1.01368780t}{T} + \cdots \\ \cdots - P\frac{0.209753235}{T}e^{-\frac{0.209753235t}{T}}\sin\frac{3.10498364t}{T} + \cdots \\ \cdots + P\frac{3.10498364}{T}e^{-\frac{0.209753235t}{T}}\cos\frac{3.10498364t}{T} + \cdots \\ \cdots - Q\frac{0.209753235}{T}e^{-\frac{0.209753235t}{T}}\cos\frac{3.10498364t}{T} + \cdots \\ \cdots - Q\frac{3.10498364}{T}e^{-\frac{0.209753235t}{T}}\sin\frac{3.10498364t}{T} \end{bmatrix}$$

$$\rightarrow \quad \frac{dI_2}{dt}(0) = G\frac{1.01368780}{T} - H\frac{0.790246764}{T} + P\frac{3.10498364}{T} - Q\frac{0.209753235}{T}$$

$$\rightarrow \quad 0 = 1.01368780G - 0.790246764H + 3.10498364P - 0.209753235Q \qquad (I.C.\#3)$$

The situation with inductor $L_1$ is different. Since there will initially be no current flowing through this inductor, there will initially be no voltage drop over the resistor $R$. $L_2$ will face the full voltage drop over capacitor $C_1$. Since the voltage drop over $L_1$ can be written in terms of the derivative of the current $I_1 + I_2$ flowing through it, we can write:

$$V_{L1}(t) = L\frac{d(I_1 + I_2)}{dt}$$

$$= L\frac{d}{dt}\left\{ \begin{array}{l} Ge^{-\frac{0.790246764t}{T}}\left[ \begin{matrix} (1 + 0.899231748)\sin\frac{1.01368780t}{T} + \cdots \\ \cdots - 0.400531752\cos\frac{1.01368780t}{T} \end{matrix} \right] + \cdots \\ \cdots + He^{-\frac{0.790246764t}{T}}\left[ \begin{matrix} (1 + 0.899231748)\cos\frac{1.01368780t}{T} + \cdots \\ \cdots + 0.400531752\sin\frac{1.01368780t}{T} \end{matrix} \right] + \cdots \\ \cdots + Pe^{-\frac{0.209753235t}{T}}\left[ \begin{matrix} (1 - 1.399231746)\sin\frac{3.10498364t}{T} + \cdots \\ \cdots - 0.325640182\cos\frac{3.10498364t}{T} \end{matrix} \right] + \cdots \\ \cdots + Qe^{-\frac{0.209753235t}{T}}\left[ \begin{matrix} (1 - 1.399231746)\cos\frac{3.10498364t}{T} + \cdots \\ \cdots + 0.325640182\sin\frac{3.10498364t}{T} \end{matrix} \right] \end{array} \right\}$$

and, continuing:

$$\frac{V_{L1}(t)}{L} = \left\{ \begin{array}{c} -G1.899231748\frac{0.790246764}{T}e^{-\frac{0.790246764t}{T}}\sin\frac{1.01368780t}{T}+\cdots \\ \cdots+G1.899231748\frac{1.01368780}{T}e^{-\frac{0.790246764t}{T}}\cos\frac{1.01368780t}{T}+\cdots \\ \cdots+G0.400531752\frac{0.790246764}{T}e^{-\frac{0.790246764t}{T}}\cos\frac{1.01368780t}{T}+\cdots \\ \cdots+G0.400531752\frac{1.01368780}{T}e^{-\frac{0.790246764t}{T}}\sin\frac{1.01368780t}{T}+\cdots \\ \cdots-H1.899231748\frac{0.790246764}{T}e^{-\frac{0.790246764t}{T}}\cos\frac{1.01368780t}{T}+\cdots \\ \cdots-H1.899231748\frac{1.01368780}{T}e^{-\frac{0.790246764t}{T}}\sin\frac{1.01368780t}{T}+\cdots \\ \cdots-H0.400531752\frac{0.790246764}{T}e^{-\frac{0.790246764t}{T}}\sin\frac{1.01368780t}{T}+\cdots \\ \cdots+H0.400531752\frac{1.01368780}{T}e^{-\frac{0.790246764t}{T}}\cos\frac{1.01368780t}{T}+\cdots \\ \cdots+P0.399231746\frac{0.209753235}{T}e^{-\frac{0.209753235t}{T}}\sin\frac{3.10498364t}{T}+\cdots \\ \cdots-P0.399231746\frac{3.10498364}{T}e^{-\frac{0.209753235t}{T}}\cos\frac{3.10498364t}{T}+\cdots \\ \cdots+P0.325640182\frac{0.209753235}{T}e^{-\frac{0.209753235t}{T}}\cos\frac{3.10498364t}{T}+\cdots \\ \cdots+P0.325640182\frac{3.10498364}{T}e^{-\frac{0.209753235t}{T}}\sin\frac{3.10498364t}{T}+\cdots \\ \cdots+Q0.399231746\frac{0.209753235}{T}e^{-\frac{0.209753235t}{T}}\cos\frac{3.10498364t}{T}+\cdots \\ \cdots+Q0.399231746\frac{3.10498364}{T}e^{-\frac{0.209753235t}{T}}\sin\frac{3.10498364t}{T}+\cdots \\ \cdots-Q0.325640182\frac{0.209753235}{T}e^{-\frac{0.209753235t}{T}}\sin\frac{3.10498364t}{T}+\cdots \\ \cdots+Q0.325640182\frac{3.10498364}{T}e^{-\frac{0.209753235t}{T}}\cos\frac{3.10498364t}{T} \end{array} \right\}$$

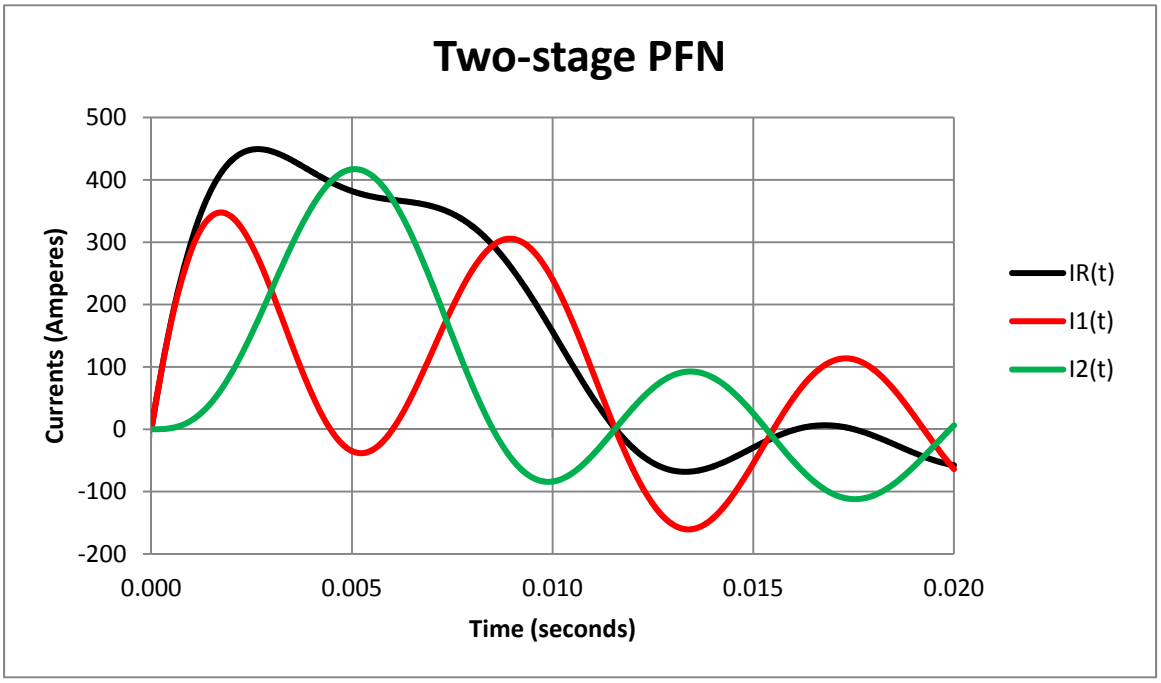
$$\rightarrow \quad \frac{V_C(0)}{L} = \frac{1}{T}\left[\begin{array}{c} G(1.899231748\times 1.01368780+0.400531752\times 0.790246764)+\cdots \\ \cdots+H(-1.899231748\times 0.790246764+0.400531752\times 1.01368780)+\cdots \\ \cdots+P(-0.399231746\times 3.10498364+0.325640182\times 0.209753235)+\cdots \\ \cdots+Q(0.399231746\times 0.209753235+0.325640182\times 3.10498364) \end{array}\right]$$

$$\rightarrow \quad \frac{V_C(0)T}{L} = 2.241746973G-1.094847592H-1.171303958P+1.094847588Q \qquad (I.C.\#4)$$

The initial conditions have given four equations in the four unknown constants. Their joint solution is the following:

$$\left.\begin{array}{ll} G & = 0.43338309\,V_C(0)T/L \\ H & = 0.05698549994\,V_C(0)T/L \\ P & = -0.1308333602\,V_C(0)T/L \\ Q & = -0.05698549994\,V_C(0)T/L \end{array}\right\} \quad (23)$$

This is the general solution for a second-order, or two-stage, pulse forming network. We have not specified the time or any of the component values. All we specified is that the capacitors be the same and that the inductors be the same, and that the ratio of their values be in accordance with Equation . The following graph shows the results if we pick a load resistance of and a pulse duration of . This sets the values of the capacitors and inductors to and , respectively.

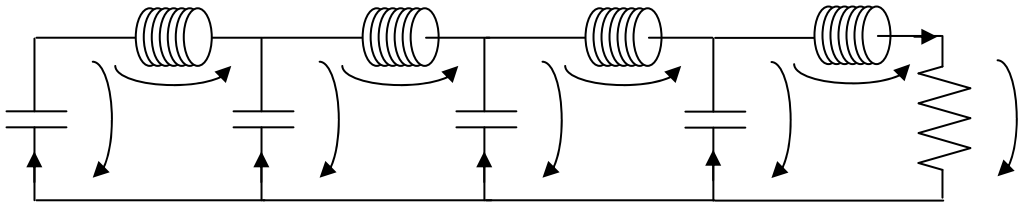



The black waveform is the current flowing through the resistor. One can actually see a square wave taking shape. One can also see the contortions the circuit makes trying to squeeze two sinusoidal currents into the required shape. It is clear that the two capacitors do this by taking turns supplying current. The first capacitor, , whose current is shown in red, goes first. As its first pulse comes to an end, the second capacitor, , whose current is shown in green, takes over. And do on, until the currents from the two of them pretty much cancel each other out.

To produce the graph above, I set the initial voltage on the capacitors to . Since the load resistor has a value of , the nominal current would be . It looks like the square wave the circuit is trying to develop has a current equal to one-half of that nominal current. It also looks like the circuit is aiming towards a pulse length of , or twice the specified value of .

**The N-stage network (where N = 4 is used as an example)**

The procedure for adding stages to the network becomes pretty clear by the time we have four stages. This is shown in the following diagram.



As before, all of the capacitors are the same and all of the inductors are the same. We are going to set the ratio of their values like we did before, as:

$$\left.\begin{aligned} C &= \frac{T}{NR} \\ L &= \frac{RT}{N} \end{aligned}\right\} \quad (24)$$

where the time interval $T$ is one-half of the duration of the square wave pulse. Where possible, I am going to use $N$ for the number of stages, and substitute $N = 4$ only as a numerical example. We can define resistor-capacitor and resistor-inductor time-constants like we did before, thus:

$$\left.\begin{aligned} \tau_{RC} &= RC = R\frac{T}{NR} = \tfrac{1}{N}T \\ \tau_{RL} &= \frac{L}{R} = \frac{1}{R}\frac{RT}{N} = \tfrac{1}{N}T \end{aligned}\right\} \quad (25)$$

We will need the products $RC$ and $LC$, which are:

$$\left.\begin{aligned} RC &= \tfrac{1}{N}T \\ LC &= \tfrac{1}{N^2}T^2 \end{aligned}\right\} \quad (26)$$

We will write down the circuit equations in stages. The first (left-most) stage, or loop, contains three components, each of which has its own voltage-current characteristic. In addition, the sum of the voltages around the loop must be zero. This gives four circuit equations.

$$\left.\begin{aligned} R\text{:} \quad & V_R = R(I_1 + I_2 + I_3 + I_4) \\ C_1\text{:} \quad & V_{C1} = V_{C1}(0) - \frac{1}{C}\int_{\tau=0}^{\tau=t} I_1 d\tau \\ L_1\text{:} \quad & V_{L1} = L\frac{d(I_1 + I_2 + I_3 + I_4)}{dt} \\ \text{Loop1:} \quad & V_{C1} = V_R + V_{L1} \end{aligned}\right\} \quad (27 - \text{Stage 1})$$

The second stage adds two more components and another loop equation, so we get three more circuit equations.

$$\left.\begin{aligned} C_2\text{:} \quad & V_{C2} = V_{C2}(0) - \frac{1}{C}\int_{\tau=0}^{\tau=t} I_2 d\tau \\ L_2\text{:} \quad & V_{L2} = L\frac{d(I_2 + I_3 + I_4)}{dt} \\ \text{Loop2:} \quad & V_{C2} = V_{C1} + V_{L2} \end{aligned}\right\} \quad (27 - \text{Stage 2})$$

The third stage adds another two components and another loop equation, so we get three more circuit equations:

$$\left.\begin{array}{ll} C_3: & V_{C3} = V_{C3}(0) - \dfrac{1}{C}\displaystyle\int_{\tau=0}^{\tau=t} I_3 d\tau \\ L_3: & V_{L3} = L\dfrac{d(I_3 + I_4)}{dt} \\ \text{Loop3:} & V_{C3} = V_{C2} + V_{L3} \end{array}\right\} \quad (27 - \text{Stage 3})$$

The fourth stage (and, incidentally, every stage added thereafter) adds two more voltage-current characteristics and another loop equation, being:

$$\left.\begin{array}{ll} C_4: & V_{C4} = V_{C4}(0) - \dfrac{1}{C}\displaystyle\int_{\tau=0}^{\tau=t} I_4 d\tau \\ L_4: & V_{L4} = L\dfrac{dI_4}{dt} \\ \text{Loop4:} & V_{C4} = V_{C3} + V_{L4} \end{array}\right\} \quad (27 - \text{Stage 4})$$

So, a pulse forming network with $N$ stages will always have $3N + 1$ circuit equations, where the extra one arises from the load resistor in the first stage. We can immediately reduce this to $2N + 1$ equations by substituting the inductor's voltage $V_{LN}$ for each stage into the loop equation for that stage. At the risk of going too slowly, let me re-state the nine equations which remain for a 4-stage network.

$$\left.\begin{array}{lll} R: & V_R = R(I_1 + I_2 + I_3 + I_4) & \text{Stage 1} \\ C_1: & V_{C1} = V_{C1}(0) - \dfrac{1}{C}\displaystyle\int_{\tau=0}^{\tau=t} I_1 d\tau & \text{Stage 1} \\ \text{Loop1:} & V_{C1} = V_R + L\dfrac{d(I_1 + I_2 + I_3 + I_4)}{dt} & \text{Stage 1} \\ C_2: & V_{C2} = V_{C2}(0) - \dfrac{1}{C}\displaystyle\int_{\tau=0}^{\tau=t} I_2 d\tau & \text{Stage 2} \\ \text{Loop2:} & V_{C2} = V_{C1} + L\dfrac{d(I_2 + I_3 + I_4)}{dt} & \text{Stage 2} \\ C_3: & V_{C3} = V_{C3}(0) - \dfrac{1}{C}\displaystyle\int_{\tau=0}^{\tau=t} I_3 d\tau & \text{Stage 3} \\ \text{Loop3:} & V_{C3} = V_{C2} + L\dfrac{d(I_3 + I_4)}{dt} & \text{Stage 3} \\ C_4: & V_{C4} = V_{C4}(0) - \dfrac{1}{C}\displaystyle\int_{\tau=0}^{\tau=t} I_4 d\tau & \text{Stage 4} \\ \text{Loop4:} & V_{C4} = V_{C3} + L\dfrac{dI_4}{dt} & \text{Stage 4} \end{array}\right\} \quad (28)$$

In the next step, the number of circuit equations will be reduced to $N + 1$, by substituting the capacitor's voltage $V_{CN}$ for each stage into the loop equation for that stage and into the loop equation for the following stage, if any. In the 4-stage case, this gives:

$$
\left.\begin{array}{lll}
R: & V_R = R(I_1 + I_2 + I_3 + I_4) & \text{Stage 1} \\
\text{Loop1:} & V_{C1}(0) - \frac{1}{C}\int_{\tau=0}^{\tau=t} I_1 d\tau = V_R + L\frac{d(I_1 + I_2 + I_3 + I_4)}{dt} & \text{Stage 1} \\
\text{Loop2:} & V_{C2}(0) - \frac{1}{C}\int_{\tau=0}^{\tau=t} I_2 d\tau = V_{C1}(0) - \frac{1}{C}\int_{\tau=0}^{\tau=t} I_1 d\tau + L\frac{d(I_2 + I_3 + I_4)}{dt} & \text{Stage 2} \\
\text{Loop3:} & V_{C3}(0) - \frac{1}{C}\int_{\tau=0}^{\tau=t} I_3 d\tau = V_{C2}(0) - \frac{1}{C}\int_{\tau=0}^{\tau=t} I_2 d\tau + L\frac{d(I_3 + I_4)}{dt} & \text{Stage 3} \\
\text{Loop4:} & V_{C4}(0) - \frac{1}{C}\int_{\tau=0}^{\tau=t} I_4 d\tau = V_{C3}(0) - \frac{1}{C}\int_{\tau=0}^{\tau=t} I_3 d\tau + L\frac{dI_4}{dt} & \text{Stage 4}
\end{array}\right\} \quad (29)
$$

The number of circuit equations can be reduced by one more, by substituting the resistor's voltage $V_R$ into the only place where it appears, in the loop equation for the first stage. In addition, let us take the time-derivative of all the equations. We get:

$$
\left.\begin{array}{lll}
\text{Loop1:} & -\frac{I_1}{C} = R\frac{d(I_1 + I_2 + I_3 + I_4)}{dt} + L\frac{d^2(I_1 + I_2 + I_3 + I_4)}{dt^2} & \text{Stage 1} \\
\text{Loop2:} & -\frac{I_2}{C} = -\frac{I_1}{C} + L\frac{d^2(I_2 + I_3 + I_4)}{dt^2} & \text{Stage 2} \\
\text{Loop3:} & -\frac{I_3}{C} = -\frac{I_2}{C} + L\frac{d^2(I_3 + I_4)}{dt^2} & \text{Stage 3} \\
\text{Loop4:} & -\frac{I_4}{C} = -\frac{I_3}{C} + L\frac{d^2 I_4}{dt^2} & \text{Stage 4}
\end{array}\right\} \quad (30)
$$

Multiplying through by $C$ and then replacing $RC$ and $LC$ with their equivalents in $T$ gives:

$$
\left.\begin{array}{lll}
\text{Loop1:} & \frac{T^2}{N^2}\frac{d^2(I_1 + I_2 + I_3 + I_4)}{dt^2} + \frac{T}{N}\frac{d(I_1 + I_2 + I_3 + I_4)}{dt} + I_1 = 0 & \text{Stage 1} \\
\text{Loop2:} & I_1 = I_2 + \frac{T^2}{N^2}\frac{d^2(I_2 + I_3 + I_4)}{dt^2} & \text{Stage 2} \\
\text{Loop3:} & I_2 = I_3 + \frac{T^2}{N^2}\frac{d^2(I_3 + I_4)}{dt^2} & \text{Stage 3} \\
\text{Loop4:} & I_3 = I_4 + \frac{T^2}{N^2}\frac{d^2 I_4}{dt^2} & \text{Stage 4}
\end{array}\right\} \quad (31)
$$

We have reduced the number of equations to the number of loops, or stages. Furthermore, it should be clear how the progression would be continued if there were additional stages. The last stage will always look like the last equation. For each preceding stage, the current from the following stage is added to the sum whose second derivative is taken. When one arrives at the first stage, it will always look like the first equation.

This is a good time to talk about initial conditions. There will always be one initial condition per capacitor and inductor. Because there is one capacitor and one inductor per stage, an $N$-stage pulse forming network will always have $2N$ initial conditions. Let us write down the initial conditions in two

groups. Assuming that the power supply starts from a condition of rest, all of the currents will initially be zero. This will provide $N$ initial conditions, one for the current through each capacitor.

The second group involves the initial voltages over the inductors. Assuming that all capacitors are initially charged up to the same voltage, the voltage drops over all of the inductors will be zero, except for the inductor in the first stage. The inductor in the first stage will face the full voltage drop over the capacitor in the first stage. Since the voltage drop over an inductor is proportional to the derivative of the current flowing through it, there will now be $N$ more initial conditions, one for each inductor in the network. We can summarize the $2N$ initial conditions as follows:

$$\left.\begin{array}{ll} I_4(0) = 0 & C_4 \text{ current} \\ I_3(0) = 0 & C_3 \text{ current} \\ I_2(0) = 0 & C_2 \text{ current} \\ I_1(0) = 0 & C_1 \text{ current} \\ \dfrac{dI_4}{dt}(0) = 0 & L_4 \text{ voltage} \\ \dfrac{d(I_3 + I_4)}{dt}(0) = 0 \;\rightarrow\; \dfrac{dI_3}{dt}(0) = 0 & L_3 \text{ voltage} \\ \dfrac{d(I_2 + I_3 + I_4)}{dt}(0) = 0 \;\rightarrow\; \dfrac{dI_2}{dt}(0) = 0 & L_2 \text{ voltage} \\ \dfrac{d(I_1 + I_2 + I_3 + I_4)}{dt} = \dfrac{V_C(0)}{L} \;\rightarrow\; \dfrac{dI_1}{dt}(0) = \dfrac{V_C(0)}{L} & L_1 \text{ voltage} \end{array}\right\} \quad (32)$$

The fact that there are $2N$ initial conditions is a sure sign that the underlying differential equation will be of the $2N^{\text{th}}$ order and that the solution for each current will have $2N$ terms. What we want to do now is to determine if there is a systematic way to solve this system of equations.

One very important observation which we made for the two-stage network is that, if the solution for one of the currents has the form $Ae^{\alpha t}$, then the corresponding solution for all the other waveforms will have the same time-dependence, that is, the same rate of change $\alpha$. Therefore, rather than going to the effort of trying to combine the $N$ loop equations, let us propose solutions for the $N$ currents, being:

$$\left.\begin{array}{l} I_1(t) = A_1 e^{\alpha t} \\ I_2(t) = A_2 e^{\alpha t} \\ I_3(t) = A_3 e^{\alpha t} \\ I_4(t) = A_4 e^{\alpha t} \end{array}\right\} \quad (33)$$

Substituting these forms and their derivatives into the loop equations gives:

$$\left.\begin{array}{ll} \text{Loop1:} & \dfrac{T^2}{N^2}\alpha^2(A_1 + A_2 + A_3 + A_4)e^{\alpha t} + \dfrac{T}{N}\alpha(A_1 + A_2 + A_3 + A_4)e^{\alpha t} + A_1 e^{\alpha t} = 0 \\ \text{Loop2:} & A_1 e^{\alpha t} = A_2 e^{\alpha t} + \dfrac{T^2}{N^2}\alpha^2(A_2 + A_3 + A_4)e^{\alpha t} \\ \text{Loop3:} & A_2 e^{\alpha t} = A_3 e^{\alpha t} + \dfrac{T^2}{N^2}\alpha^2(A_3 + A_4)e^{\alpha t} \\ \text{Loop4:} & A_3 e^{\alpha t} = A_4 e^{\alpha t} + \dfrac{T^2}{N^2}\alpha^2 A_4 e^{\alpha t} \end{array}\right\} \quad (34)$$

Replacing the term $\alpha T/N$ with the variable $x$ (this is not exactly what we did for the two-stage network, where we replaced $\alpha T$ with the variable $x$, and left the factor $1/N$ as part of the coefficients of the polynomial) and eliminating the common exponential factor, gives:

$$\left.\begin{array}{ll} \text{Loop1:} & x^2(A_1+A_2+A_3+A_4)+x(A_1+A_2+A_3+A_4)+A_1=0 \\ \text{Loop2:} & A_1=A_2+x^2(A_2+A_3+A_4) \\ \text{Loop3:} & A_2=A_3+x^2(A_3+A_4) \\ \text{Loop4:} & A_3=A_4+x^2A_4 \end{array}\right\} \quad (35)$$

It is useful to re-order the terms in these equations, for all of the loops except the first, as follows.

$$\left.\begin{array}{ll} \text{Loop4:} & A_3=(1+x^2)A_4 \\ \text{Loop3:} & A_2=(1+x^2)A_3+x^2A_4 \\ \text{Loop2:} & A_1=(1+x^2)A_2+x^2(A_3+A_4) \end{array}\right\} \quad (36)$$

This system of equations can easily be scaled up if there are additional stages in the network. For example, the beginning of the list for a 17-stage network can be written down by inspection as:

$$\begin{array}{ll} \text{Loop17:} & A_{16}=(1+x^2)A_{17} \\ \text{Loop16:} & A_{15}=(1+x^2)A_{16}+x^2A_{17} \\ \text{Loop15:} & A_{14}=(1+x^2)A_{15}+x^2(A_{16}+A_{17}) \\ \text{Loop14:} & A_{13}=(1+x^2)A_{14}+x^2(A_{15}+A_{16}+A_{17}) \\ \text{Loop13:} & A_{12}=(1+x^2)A_{13}+x^2(A_{14}+A_{15}+A_{16}+A_{17}) \\ & \text{and so forth} \end{array}$$

These expansions can be combined recursively, starting from the bottom. This will result in a set of polynomials. There will be one polynomial for each stage in the network (other than the first stage). Furthermore, each polynomial will have exactly the same factor, being the constant $A_N$ for the current in the last stage of the network. The algebra involved in expanding the recursion is considerable, but straight-forward. The following table sets out the polynomials in $x$ for various numbers of stages. Since one always begins with the final stage in a network, these polynomials are always the same.

$$\left.\begin{array}{l} A_N = A_N \\ A_{N-1}=(x^2+1)A_N \\ A_{N-2}=(x^4+3x^2+1)A_N \\ A_{N-3}=(x^6+5x^4+6x^2+1)A_N \\ A_{N-4}=(x^8+7x^6+15x^4+10x^2+1)A_N \\ A_{N-5}=(x^{10}+9x^8+28x^6+35x^4+15x^2+1)A_N \\ A_{N-6}=(x^{12}+11x^{10}+45x^8+84x^6+70x^4+21x^2+1)A_N \\ A_{N-7}=(x^{14}+13x^{12}+66x^{10}+165x^8+210x^6+126x^4+28x^2+1)A_N \\ A_{N-8}=(x^{16}+15x^{14}+91x^{12}+286x^{10}+495x^8+462x^6+210x^4+36x^2+1)A_N \end{array}\right\} \quad (37)$$

Let me reiterate the interpretation of the polynomials in Equation (37). If one is designing a pulse forming network with $M$ stages, then one uses the first $M$ of these polynomials. Then, if a solution for the current in the last stage is $I_M(t)=A_Me^{xt}$, then the corresponding current in the $i^{\text{th}}$ stage is given by $I_i(t)=A_ie^{xt}$, where the coefficient $A_i$ is the appropriate row in Equation (37). The last row in the table will give $A_1$, for the first stage.

I have attached as Appendix "A" hereto an extension of Equation (37) up to the 20th order.

Now, let us return to the full set of circuit equations. We have dealt with all of the circuit equations except the one for the first stage. The first stage is different from the ganged stages which follow it. In a sense though, it is even simpler. For an $N$-stage network, the equation for the first stage has the following form:

$$\text{Loop1:} \quad x^2\left(\sum_{i=1}^{N} A_i\right) + x\left(\sum_{i=1}^{N} A_i\right) + A_1 = 0 \quad (38)$$

Since the process of recursion gives expressions for all of the $A_i$'s as products of polynomials in $x$ and the final-stage coefficient $A_N$, their substitution into Equation (38) will result in a polynomial with the same form. Furthermore, their substitution into Equation (37) is really only a matter of adding up the polynomials set out in the table above. The results from doing this are the following:

$$\left.\begin{array}{ll} \text{1 stage:} & 0 = (x^2 + x + 1)A_1 \\ \text{2 stage:} & 0 = (x^4 + x^3 + 2x^2 + 2x + 1)A_2 \\ \text{3 stage:} & 0 = (x^6 + x^5 + 5x^4 + 4x^3 + 6x^2 + 3x + 1)A_3 \\ \text{4 stage:} & 0 = (x^8 + x^7 + 7x^6 + 8x^5 + 15x^4 + 10x^3 + 10x^2 + 4x + 1)A_4 \\ \text{5 stage:} & 0 = (x^{10} + x^9 + 9x^8 + 8x^7 + 28x^6 + 21x^5 + 35x^4 + 20x^3 + 15x^2 + 5x + 1)A_5 \end{array}\right\} \quad (39)$$

For an $N$-stage network, the corresponding line in this table is the characteristic equation of the differential equation for the current in the final $N^{\text{th}}$ stage. This table is easily expanded to higher orders – the page is just not wide enough to show the higher-order polynomials on single lines. I have attached as Appendix "B" hereto an extension of Equation (39) up to the 20th order

For the fourth-order network we are using as an example, it is the fourth line of Equation (39) which contains the characteristic equation. The next step in the process is to find the roots of this characteristic equation, which roots will describe the time-dependence of current $I_4$. There will be eight roots, corresponding to eight solutions. The roots will consist of four conjugate pairs. (Because of the ratio between the capacitances and the inductances which is built into the configuration, there will never be a pair of roots which are strictly real.

Finding roots of a polynomial is a straight-forward operation. I have attached as Appendix "C" hereto a schedule setting out the roots for networks of up to the 10th order. In addition, I have attached as Appendix "D" hereto a description of a procedure to find these roots, which is particularly useful for even-ordered polynomials with real coefficients. In any event, the eight roots for the fourth-order network are the following.

$$\left.\begin{array}{cc} \text{Root} & \text{Root } x \\ 1 & -0.27222 + j0.278653 \\ 2 & -0.27222 - j0.278653 \\ 3 & -0.14779 + j0.913548 \\ 4 & -0.14779 - j0.913548 \\ 5 & -0.06403 + j1.484789 \\ 6 & -0.06403 - j1.484789 \\ 7 & -0.01595 + j1.866410 \\ 8 & -0.01595 - j1.866410 \end{array}\right\} \quad (40)$$

Since these roots describe the time-dependence of current $I_4(t)$, we can write:

$$I_4(t) = \begin{bmatrix} e^{-0.27222t}\left(A_{41}e^{+j0.278653t} + A_{42}e^{-j0.278653t}\right) + \cdots \\ \cdots + e^{-0.14779t}\left(A_{43}e^{+j0.913548t} + A_{44}e^{-j0.913548t}\right) + \cdots \\ \cdots + e^{-0.06403t}\left(A_{45}e^{+j1.484789t} + A_{46}e^{-j1.484789t}\right) + \cdots \\ \cdots + e^{-0.01595t}\left(A_{47}e^{+j1.866410t} + A_{48}e^{-j1.866410t}\right) \end{bmatrix} \quad (41)$$

For the purpose of what follows, it is simpler to leave the exponentials with imaginary arguments as they are rather than convert them into their equivalent sinusoidal forms. I am going to try to be consistent in the identification of the coefficients for the various currents. The first subscript $m$ identifies the loop in which the current occurs and the second subscript $n$ identifies the particular root in the ordered list of roots.

The first thing we are going to want to do is to find the waveforms of the other currents. For this purpose, we can go back to the expanded polynomials in Table (37). Since we are looking at a fourth-order network, the last stage has $N = 4$. The second row in the table lets us calculate $I_3(t)$ as $A_3 = (x^2 + 1)A_4$. The third row in the table lets us calculate $I_2(t)$ as $A_2 = (x^4 + 3x^2 + 1)A_N$, and so on. Noting that the roots listed in Equation (40) are, in fact, the values of $x$, we get for $I_3(t)$:

$$\left.\begin{aligned} A_{31} &= (x_1^2 + 1)A_{41} = [(-0.27222 + j0.278653)^2 + 1]A_{41} \\ A_{32} &= (x_2^2 + 1)A_{42} = [(-0.27222 - j0.278653)^2 + 1]A_{42} \\ A_{33} &= (x_3^2 + 1)A_{43} = [(-0.14779 + j0.913548)^2 + 1]A_{43} \\ A_{34} &= (x_4^2 + 1)A_{44} = [(-0.14779 - j0.913548)^2 + 1]A_{44} \\ A_{35} &= \left(x_5^2 + 1\right)A_{45} = [(-0.06403 + j1.484789)^2 + 1]A_{45} \\ A_{36} &= (x_6^2 + 1)A_{46} = [(-0.06403 - j1.484789)^2 + 1]A_{46} \\ A_{37} &= (x_7^2 + 1)A_{47} = [(-0.01595 + j1.866410)^2 + 1]A_{47} \\ A_{38} &= (x_8^2 + 1)A_{48} = [(-0.01595 - j1.866410)^2 + 1]A_{48} \end{aligned}\right\} \quad (42)$$

Current $I_4(t)$ is the sum of eight terms, with one term corresponding to each root, and with one coefficient for each term. Similarly, current $I_3(t)$ is the sum of eight terms, with one term corresponding to each root, and with one coefficient for each term. Each of the coefficients in $I_3(t)$ is a complex number multiplied by the corresponding coefficient of $I_4(t)$.

We next use the third row of Table (37) to get the coefficients of the eight terms whose sum is current $I_2(t)$. From the third row, we know that $A_2 = (x^4 + 3x^2 + 1)A_4$, so that:

$$\left.\begin{aligned} A_{21} &= (x_1^4 + 3x_1^2 + 1)A_{41} \\ &= [(-0.06403 + j1.484789)^4 + 3(-0.06403 + j1.484789)^2 + 1]A_{41} \\ A_{22} &= (x_2^4 + 3x_2^2 + 1)A_{42} \\ &= [(-0.06403 - j1.484789)^4 + 3(-0.06403 - j1.484789)^2 + 1]A_{42} \\ A_{23} &= (x_3^4 + 3x_3^2 + 1)A_{43} \\ &= [(-0.14779 + j0.913548)^4 + 3(-0.14779 + j0.913548)^2 + 1]A_{43} \\ A_{24} &= (x_4^4 + 3x_4^2 + 1)A_{44} \\ &= [(-0.14779 - j0.913548)^4 + 3(-0.14779 - j0.913548)^2 + 1]A_{44} \end{aligned}\right\} \quad (43 - \text{part A})$$

$$\left.\begin{aligned}
A_{25} &= \left(x_5^4 + 3x_5^2 + 1\right)A_{45} \\
&= [(-0.06403 + j1.484789)^4 + 3(-0.06403 + j1.484789)^2 + 1]A_{45} \\
A_{26} &= (x_6^4 + 3x_6^2 + 1)A_{46} \\
&= [(-0.06403 - j1.484789)^4 + 3(-0.06403 - j1.484789)^2 + 1]A_{46} \\
A_{27} &= (x_7^4 + 3x_7^2 + 1)A_{47} \\
&= [(-0.01595 + j1.866410)^4 + 3(-0.01595 + j1.866410)^2 + 1]A_{47} \\
A_{28} &= (x_8^4 + 3x_8^2 + 1)A_{48} \\
&= [(-0.01595 - j1.866410)^4 + 3(-0.01595 - j1.866410)^2 + 1]A_{48}
\end{aligned}\right\} \quad (43 - \text{part B})$$

This gives us the eight coefficients need to write down the solution for $I_2(t)$. Lastly, we can tackle current $I_1(t)$ using the fourth row of Table (37), namely, $A_1 = (x^6 + 5x^4 + 6x^2 + 1)A_4$. We get:

$$\left.\begin{aligned}
A_{11} &= (x_1^6 + 5x_1^4 + 6x_1^2 + 1)A_{41} \\
&= \begin{bmatrix}(-0.06403 + j1.484789)^6 + 5(-0.06403 + j1.484789)^4 + \cdots \\ +5(-0.06403 + j1.484789)^2 + 1\end{bmatrix} A_{41} \\
A_{12} &= (x_2^6 + 5x_2^4 + 6x_2^2 + 1)A_{42} \\
&= \begin{bmatrix}(-0.06403 - j1.484789)^6 + 5(-0.06403 - j1.484789)^4 + \cdots \\ +5(-0.06403 - j1.484789)^2 + 1\end{bmatrix} A_{42} \\
A_{13} &= (x_3^6 + 5x_3^4 + 6x_3^2 + 1)A_{43} \\
&= \begin{bmatrix}(-0.14779 + j0.913548)^6 + 5(-0.14779 + j0.913548)^4 + \cdots \\ +6(-0.14779 + j0.913548)^2 + 1\end{bmatrix} A_{43} \\
A_{14} &= (x_4^6 + 5x_4^4 + 6x_4^2 + 1)A_{44} \\
&= \begin{bmatrix}(-0.14779 - j0.913548)^6 + 5(-0.14779 - j0.913548)^4 + \cdots \\ +6(-0.14779 - j0.913548)^2 + 1\end{bmatrix} A_{44} \\
A_{15} &= \left(x_5^6 + 5x_5^4 + 6x_5^2 + 1\right)A_{45} \\
&= \begin{bmatrix}(-0.06403 + j1.484789)^6 + 5(-0.06403 + j1.484789)^4 + \cdots \\ +6(-0.06403 + j1.484789)^2 + 1\end{bmatrix} A_{45} \\
A_{16} &= (x_6^6 + 5x_6^4 + 6x_6^2 + 1)A_{46} \\
&= \begin{bmatrix}(-0.06403 - j1.484789)^6 + 5(-0.06403 - j1.484789)^4 + \cdots \\ +6(-0.06403 - j1.484789)^2 + 1\end{bmatrix} A_{46} \\
A_{17} &= (x_7^6 + 5x_7^4 + 6x_7^2 + 1)A_{47} \\
&= \begin{bmatrix}(-0.01595 + j1.866410)^6 + 5(-0.01595 + j1.866410)^4 + \cdots \\ +6(-0.01595 + j1.866410)^2 + 1\end{bmatrix} A_{47} \\
A_{18} &= (x_8^6 + 5x_8^4 + 6x_8^2 + 1)A_{48} \\
&= \begin{bmatrix}(-0.01595 - j1.866410)^6 + 5(-0.01595 - j1.866410)^4 + \cdots \\ +6(-0.01595 - j1.866410)^2 + 1\end{bmatrix} A_{48}
\end{aligned}\right\} \quad (44)$$

This completes the expansions for the fourth-order network. For a higher order network, one simply keeps working down Table (37) until the $2N$ coefficients for $I_1(t)$ have been computed.

Some readers may find it helpful if I express the currents using matrix algebra, as follows:

$$\begin{bmatrix} I_4(t) \\ I_3(t) \\ I_2(t) \\ I_1(t) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \frac{A_{31}}{A_{41}} & \frac{A_{32}}{A_{42}} & \frac{A_{33}}{A_{43}} & \frac{A_{34}}{A_{44}} & \frac{A_{35}}{A_{45}} & \frac{A_{36}}{A_{46}} & \frac{A_{37}}{A_{47}} & \frac{A_{38}}{A_{48}} \\ \frac{A_{21}}{A_{41}} & \frac{A_{22}}{A_{42}} & \frac{A_{23}}{A_{43}} & \frac{A_{24}}{A_{44}} & \frac{A_{25}}{A_{45}} & \frac{A_{26}}{A_{46}} & \frac{A_{27}}{A_{47}} & \frac{A_{28}}{A_{48}} \\ \frac{A_{11}}{A_{41}} & \frac{A_{12}}{A_{42}} & \frac{A_{13}}{A_{43}} & \frac{A_{14}}{A_{44}} & \frac{A_{15}}{A_{45}} & \frac{A_{16}}{A_{46}} & \frac{A_{17}}{A_{47}} & \frac{A_{18}}{A_{48}} \end{bmatrix} \begin{bmatrix} A_{41}e^{-0.27222t+j0.278653t} \\ A_{42}e^{-0.27222t-j0.278653t} \\ A_{43}e^{-0.14779t+j0.913548t} \\ A_{44}e^{-0.14779t-j0.913548t} \\ A_{45}e^{-0.06403t+j1.484789t} \\ A_{46}e^{-0.06403t-j1.484789t} \\ A_{47}e^{-0.01595t+j1.866410t} \\ A_{48}e^{-0.01595t-j1.866410t} \end{bmatrix} \quad (45)$$

What is important is that the matrix (as distinct from the two column vectors) is an array of fixed complex numbers. Each order of pulse forming network has a unique matrix. For any given order, the matrix is the same – it does not depend on the component values, the duration $T$, or, most importantly, on the initial conditions. All of the information about the waveforms and the initial conditions is contained in the last column vector. To emphasize the point, the values in the matrix are not variables. The only variables in Equation (45) are the eight unknowns which are the eight coefficients in the last column vector.

Before running off to invert this constant, albeit complex, matrix, there is something to see. Consider any pair of elements in the matrix which correspond to exponents which are complex conjugates. The pair $A_{23}$ and $A_{24}$ is shown in a box in Equation (45). This pair corresponds to the two roots $-0.14779 \pm j0.913548$. All of the elements in the matrix can be divided into such pairs, including the elements in the first row, since the complex conjugate of 1 is 1. Each such pair of elements in the matrix will always be complex conjugates. For example, compare the expressions for $A_{23}$ and $A_{24}$ from above:

$$\left.\begin{aligned} A_{23} &= [(-0.14779 + j0.913548)^4 + 3(-0.14779 + j0.913548)^2 + 1]A_{43} \\ A_{24} &= [(-0.14779 - j0.913548)^4 + 3(-0.14779 - j0.913548)^2 + 1]A_{44} \end{aligned}\right\} \quad (46)$$

The squares of two complex conjugate numbers are themselves conjugates: $(a \pm jb)^2 = (a^2 - b^2) \pm 2jab$. It follows that the fourth power of two complex conjugates are also conjugates. Any even power of two complex conjugates will be conjugates. All of the expressions which expand the $A$'s in terms of those of the last stage involve even powers of the roots $x$ – see the expansions in Equation (37). This is important. In due course, we will see that this pairing is what allows the solution matrix to be divided into separate cosine and sine parts, each of which is entirely real.

For convenience, I am going to use the symbol $\widetilde{M}_N$ for the constant matrix for an $N^{\text{th}}$-order network. Let us now apply the first set of initial conditions – that all four currents are initially zero. Setting $t = 0$ in Equation (45) gives the following identity:

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \widetilde{M}_4 \begin{bmatrix} A_{41} \\ A_{42} \\ A_{43} \\ A_{44} \\ A_{45} \\ A_{46} \\ A_{47} \\ A_{48} \end{bmatrix} \quad (\text{I. C. set \#1})$$

This constitutes four equations in the eight unknowns contained in the last column vector. Next, the derivatives of all four currents can be written down from Equation (45) as:

$$\begin{bmatrix} \frac{dI_4(t)}{dt} \\ \frac{dI_3(t)}{dt} \\ \frac{dI_2(t)}{dt} \\ \frac{dI_1(t)}{dt} \end{bmatrix} = \widetilde{M}_4 \begin{bmatrix} A_{41}(-0.27222 + j0.278653)e^{-0.27222t+j0.278653t} \\ A_{42}(-0.27222 - j0.278653)e^{-0.27222t-j0.278653t} \\ A_{43}(-0.14779 + j0.913548)e^{-0.14779t+j0.913548t} \\ A_{44}(-0.14779 - j0.913548)e^{-0.14779t-j0.913548t} \\ A_{45}(-0.06403 + j1.484789)e^{-0.06403t+j1.484789t} \\ A_{46}(-0.06403 - j1.484789)e^{-0.06403t-j1.484789t} \\ A_{47}(-0.01595 + j1.866410)e^{-0.01595t+j1.866410t} \\ A_{48}(-0.01595 - j1.866410)e^{-0.01595t-j1.866410t} \end{bmatrix} \quad (47)$$

When we evaluate these derivatives at time $t = 0$, we get:

$$\begin{bmatrix} \frac{dI_4(t)}{dt}(0) \\ \frac{dI_3(t)}{dt}(0) \\ \frac{dI_2(t)}{dt}(0) \\ \frac{dI_1(t)}{dt}(0) \end{bmatrix} = \widetilde{M}_4 \begin{bmatrix} A_{41}(-0.27222 + j0.278653) \\ A_{42}(-0.27222 - j0.278653) \\ A_{43}(-0.14779 + j0.913548) \\ A_{44}(-0.14779 - j0.913548) \\ A_{45}(-0.06403 + j1.484789) \\ A_{46}(-0.06403 - j1.484789) \\ A_{47}(-0.01595 + j1.866410) \\ A_{48}(-0.01595 - j1.866410) \end{bmatrix} \quad (48)$$

Now, let us apply the second set of initial conditions, which depend on derivatives of the current. For convenience, I will repeat those initial conditions, which we looked at above.

$$\left.\begin{array}{ll} \frac{dI_4}{dt}(0) = 0 & L_4 \text{ voltage} \\ \frac{dI_3}{dt}(0) = 0 & L_3 \text{ voltage} \\ \frac{dI_2}{dt}(0) = 0 & L_2 \text{ voltage} \\ \frac{dI_1}{dt}(0) = \frac{V_C(0)}{L} & L_1 \text{ voltage} \end{array}\right\} \quad (32)$$

Making the substitutions into the left-hand side of Equation (48) gives another four equations in the eight unknowns.

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ \left(\frac{T}{N}\right)\frac{V_C(0)}{L} \end{bmatrix} = \widetilde{M}_4 \begin{bmatrix} A_{41}(-0.27222 + j0.278653) \\ A_{42}(-0.27222 - j0.278653) \\ A_{43}(-0.14779 + j0.913548) \\ A_{44}(-0.14779 - j0.913548) \\ A_{45}(-0.06403 + j1.484789) \\ A_{46}(-0.06403 - j1.484789) \\ A_{47}(-0.01595 + j1.866410) \\ A_{48}(-0.01595 - j1.866410) \end{bmatrix} \quad (\text{I. C. set \#2})$$

Alert readers will notice that the constant $V_C(0)/L$ found its way into Equation (I. C. set #2) in a slightly different form, as $(T/N)\,V_C(0)/L$. This arises because we changed the parameter $\alpha$ just before Equation (25). Before that point, we were using a solutions for the current of form $e^{\alpha t}$. We found it convenient to

replace $\alpha$ with a different parameter $x$, where $x$ was defined as $x = \alpha T/N$. Inverted, this means that $\alpha = Nx/T$ and the solutions for the current should be written in the time domain as $e^{Nxt/T}$. In effect, we "normalized" the time axis by dividing time $t$ by $T/N$. Ever since Equation (35), we have been using the parameter $x$ instead of the parameter $\alpha$. This change applies to the derivatives as well. The derivatives should really be transformed from the time domain to the "normalized time domain" as follows:

$$\begin{array}{ll} \text{Time domain} & \text{Normalized time domain} \\ \dfrac{d}{dt} & \begin{aligned} \dfrac{d}{dt'} &= \dfrac{d}{d\left(\dfrac{t}{N/T}\right)} \\ &= \left(\dfrac{T}{N}\right)\dfrac{d}{dt} \end{aligned} \end{array} \qquad (49)$$

Between Equations (I. C. set #1) and (I. C. set #2), we now have eight equations in the eight unknowns. If the complex numbers on the tight-hand side of Equation (I. C. set #2) look familiar, it is because they are. The eight numbers, from top to bottom, are the eight individual roots of the fourth-order polynomial. Compare them with the list in Equation (40). When writing out Equations (42) through (44), we used the symbols $x_1$, $x_2$, $x_3$, $x_4$, $x_5$, $x_6$, $x_7$ and $x_8$ for the eight roots. Using these symbols for the roots, we can combine the two sets of initial conditions, in Equations (I. C. set #1) and (I. C. set #2) as follows.

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \\ 0 \\ 0 \\ 0 \\ \left(\dfrac{T}{N}\right)\dfrac{V_C(0)}{L} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \dfrac{A_{31}}{A_{41}} & \dfrac{A_{32}}{A_{42}} & \dfrac{A_{33}}{A_{43}} & \dfrac{A_{34}}{A_{44}} & \dfrac{A_{35}}{A_{45}} & \dfrac{A_{36}}{A_{46}} & \dfrac{A_{37}}{A_{47}} & \dfrac{A_{38}}{A_{48}} \\ \dfrac{A_{21}}{A_{41}} & \dfrac{A_{22}}{A_{42}} & \dfrac{A_{23}}{A_{43}} & \dfrac{A_{24}}{A_{44}} & \dfrac{A_{25}}{A_{45}} & \dfrac{A_{26}}{A_{46}} & \dfrac{A_{27}}{A_{47}} & \dfrac{A_{28}}{A_{48}} \\ \dfrac{A_{11}}{A_{41}} & \dfrac{A_{12}}{A_{42}} & \dfrac{A_{13}}{A_{43}} & \dfrac{A_{14}}{A_{44}} & \dfrac{A_{15}}{A_{45}} & \dfrac{A_{16}}{A_{46}} & \dfrac{A_{17}}{A_{47}} & \dfrac{A_{18}}{A_{48}} \\ \\ x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 \\ \dfrac{A_{31}}{A_{41}}x_1 & \dfrac{A_{32}}{A_{42}}x_2 & \dfrac{A_{33}}{A_{43}}x_3 & \dfrac{A_{34}}{A_{44}}x_4 & \dfrac{A_{35}}{A_{45}}x_5 & \dfrac{A_{36}}{A_{46}}x_6 & \dfrac{A_{37}}{A_{47}}x_7 & \dfrac{A_{38}}{A_{48}}x_8 \\ \dfrac{A_{21}}{A_{41}}x_1 & \dfrac{A_{22}}{A_{42}}x_2 & \dfrac{A_{23}}{A_{43}}x_3 & \dfrac{A_{24}}{A_{44}}x_4 & \dfrac{A_{25}}{A_{45}}x_5 & \dfrac{A_{26}}{A_{46}}x_6 & \dfrac{A_{27}}{A_{47}}x_7 & \dfrac{A_{28}}{A_{48}}x_8 \\ \dfrac{A_{11}}{A_{41}}x_1 & \dfrac{A_{12}}{A_{42}}x_2 & \dfrac{A_{13}}{A_{43}}x_3 & \dfrac{A_{14}}{A_{44}}x_4 & \dfrac{A_{15}}{A_{45}}x_5 & \dfrac{A_{16}}{A_{46}}x_6 & \dfrac{A_{17}}{A_{47}}x_7 & \dfrac{A_{18}}{A_{48}}x_8 \end{bmatrix} \begin{bmatrix} A_{41} \\ A_{42} \\ A_{43} \\ A_{44} \\ A_{45} \\ A_{46} \\ A_{47} \\ A_{48} \end{bmatrix} \qquad (50)$$

All of the elements in this $8 \times 8$ matrix are fixed complex numbers, which applies to any fourth-order network. Fortunately we are not going to need to find the inverse of this matrix in a formal manner. It lends itself to solution by Euler's method, where each row is successively substituted into the other rows. Because the capacitors and inductors in the network are balanced, this is a very well-conditioned, or balanced, matrix. Its conditioning can be improved even further if the roots are listed in order of decreasing magnitude, as they are in Equation (40) above. That seven of the eight equations have zero on the left-hand side makes Euler's method even easier to use.

To give readers some idea of the range of elements involved, the following is the matrix for a fourth-order network.

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0.996 - j0.152 & 0.996 + j0.152 & 0.187 - j0.270 & 0.187 + j0.270 & -1.201 - j0.190 & -1.201 + j0.190 & -2.483 - j0.060 & -2.483 + j0.060 \\ 0.966 - j0.454 & 0.966 + j0.454 & -0.851 - j0.371 & -0.851 + j0.371 & -0.795 + j0.266 & -0.795 - j0.266 & 2.680 + j0.236 & 2.680 - j0.236 \\ 0.864 - j0.901 & 0.864 + j0.901 & -1.297 + j0.059 & -1.297 - j0.059 & 1.411 + j0.288 & 1.411 - j0.288 & -1.477 - j0.450 & -1.477 + j0.450 \\ -0.272 + j0.279 & -0.272 - j0.279 & -0.148 + j0.914 & -0.148 - j0.914 & -0.064 + j1.485 & -0.064 - j1.485 & -0.016 + j1.866 & -0.016 - j1.866 \\ -0.229 + j0.319 & -0.229 - j0.319 & 0.219 + j0.211 & 0.219 - j0.211 & 0.359 - j1.770 & 0.359 + j1.770 & 0.151 - j4.634 & 0.151 + j4.634 \\ -0.137 + j0.393 & -0.137 - j0..393 & 0.465 - j0.722 & 0.465 + j0.722 & -0.345 - j1.198 & -0.345 + j1.198 & -0.484 + j4.998 & -0.484 - j4.998 \\ 0.016 + j0.486 & 0.016 - j0.486 & 0.138 - j1.194 & 0.138 + j1.194 & -0.518 + j2.076 & -0.518 - j2.076 & 0.864 - j2.750 & 0.864 + j2.750 \end{bmatrix}$$

Equation (50) is easily reduced to upper triangular form using Euler's method. After this is done, Equation (50) for the fourth-order network looks like the following:

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -0.033 + j0.009 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & -0.390 + j2.667 & 1.390 + j2.667 & -0.127 + j7.241 & 1.127 + j7.241 & 0.304 + j11.468 & 0.696 + j11.468 \\ 0 & 0 & 1 & 0.314 - j0.949 & 5.884 - j2.8706 & 4.573 - j4.685 & 13.499 - j9.067 & 12.849 - j9.968 \\ 0 & 0 & 0 & 1 & -11.363 + j12.444 & -2.361 + j17.566 & -44.282 + j67.419 & -36.599 + j72.285 \\ 0 & 0 & 0 & 0 & 1 & 0.860 - j0.510 & 4.693 - j1.055 & 4.576 - j1.484 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1.020 + j1.011 & 3.300 + j1.405 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0.356 + j0.935 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.497 - j1.457 \end{bmatrix}$$

Note that the reduced matrix is associated with a column vector which contains the cumulative factor by which each row was multiplied during the triangularization process. In our case, the only relevant factor is the one for the last row. For the last row, the left-hand side value$(T/N)\,V_C(0)/L$ will be multiplied by $-0.033 + j0.009$.

Solving for the constants $A_{41}$, $A_{42}$, $A_{43}$, $A_{44}$, $A_{45}$, $A_{46}$, $A_{47}$ and $A_{48}$ is now a simple matter of back-substituting through the rows, from the last equation upwards. The first few substitutions are as follows:

$$\left.\begin{aligned} &\text{row 8:} \quad (0.497 - j1.457)A_{48} = (-0.033 + j0.009)\left(\frac{T}{N}\right)\frac{V_C(0)}{L} \\ &\rightarrow \quad A_{48} = \frac{(-0.033 + j0.009)}{(0.497 - j1.457)}\left(\frac{T}{N}\right)\frac{V_C(0)}{L} \\ &\text{row 7:} \quad A_{47} + (0.356 + j0.935)A_{48} = 0 \\ &\rightarrow \quad A_{47} = -(0.356 + j0.935)\frac{(-0.033 + j0.009)}{(0.497 - j1.457)}\left(\frac{T}{N}\right)\frac{V_C(0)}{L} \\ &\text{row 6:} \quad A_{46} + (1.020 + j1.011)A_{47} + (3.300 + j1.405)A_{48} \\ &\rightarrow \quad A_{46} = \begin{bmatrix} (1.020 + j1.011)(0.356 + j0.935)(3.300 + j1.405)\dfrac{(-0.033 + j0.009)}{(0.497 - j1.457)} + \cdots \\ \cdots - (3.300 + j1.405)\dfrac{(-0.033 + j0.009)}{(0.497 - j1.457)} \end{bmatrix}\left(\frac{T}{N}\right)\frac{V_C(0)}{L} \end{aligned}\right\} \quad (51)$$

Completing the back substitution gives the eight coefficients $A_{41}$, $A_{42}$, $A_{43}$, $A_{44}$, $A_{45}$, $A_{46}$, $A_{47}$ and $A_{48}$ which are the coefficients of the eight terms whose sum is the current $I_4(t)$ in the last loop. Note that each coefficient has $(T/N)\,V_C(0)/L$ as a factor.

The next step is to use Equations (42) through (44) to obtain the coefficients for the currents in the other loops. When this is done, all eight coefficients for all four currents will be known. For stage #$m$ as an example, the current will have the form.

$$I_m(t) = A_{m1}e^{\frac{N}{T}x_1 t} + A_{m2}e^{\frac{N}{T}x_2 t} + A_{m3}e^{\frac{N}{T}x_3 t} + A_{m4}e^{\frac{N}{T}x_4 t} + A_{m5}e^{\frac{N}{T}x_5 t} + A_{m6}e^{\frac{N}{T}x_6 t} + A_{m7}e^{\frac{N}{T}x_7 t} + A_{m8}e^{\frac{N}{T}x_8 t} \quad (52)$$

I have expressed the current in the time domain using, once again, the relationship $\alpha = Nx/T$. The coefficients $A_{m1}$, $A_{m2}$, $A_{m3}$, $A_{m4}$, $A_{m5}$, $A_{m6}$, $A_{m7}$ and $A_{m8}$ in Equation (52) are complex numbers. It is at this point that we will use the observation that each pair of roots – $x_1$ and $x_2$, $x_3$ and $x_4$, $x_5$ and $x_6$, and so forth – are complex conjugates. The two terms in $I_m(t)$ which correspond to the pair of conjugate roots can be combined to express $I_m(t)$ in terms of the cosine and sine functions. For example, consider the first pair of roots $x_1$ and $x_2$. Let us express them in the standard complex number format, so that $x_1 = a + jb$ and $x_{12} = a - jb$, where $a$ and $b$ are strictly real numbers. Similarly, let us express the two coefficients $A_{m1}$ and $A_{m2}$ in standard form, as $A_{m1} = G + jH$ and $A_{m2} = P + jQ$, where $G$, $H$, $P$ and $Q$ are also strictly real numbers. Then, the sum of the two terms in $I_m(t)$ can be written as follows:

$$\begin{aligned} A_{m1}e^{\frac{N}{T}x_1 t} + A_{m2}e^{\frac{N}{T}x_2 t} &= (G + jH)e^{\frac{N}{T}(a+jb)t} + (P + jQ)e^{\frac{N}{T}(a-jb)t} \\ &= e^{\frac{N}{T}at}\left[(G + jH)e^{j\frac{N}{T}bt} + (P + jQ)e^{-j\frac{N}{T}bt}\right] \\ &= e^{\frac{N}{T}at}\left[(G + jH)\left(\cos\frac{N}{T}bt + j\sin\frac{N}{T}bt\right) + (P + jQ)\left(\cos\frac{N}{T}bt - j\sin\frac{N}{T}bt\right)\right] \\ &= e^{\frac{N}{T}at}\left[(G + jH + P + jQ)\cos\frac{N}{T}bt + (jG - H - jP + Q)\sin\frac{N}{T}bt\right] \end{aligned} \quad (53)$$

It will be the case that $G + jH + P + jQ$ and $jG - H - jP + Q$ are strictly real. This provides a useful check of all the arithmetic which went before. If either of these factors has a non-real component, then an error has occurred.

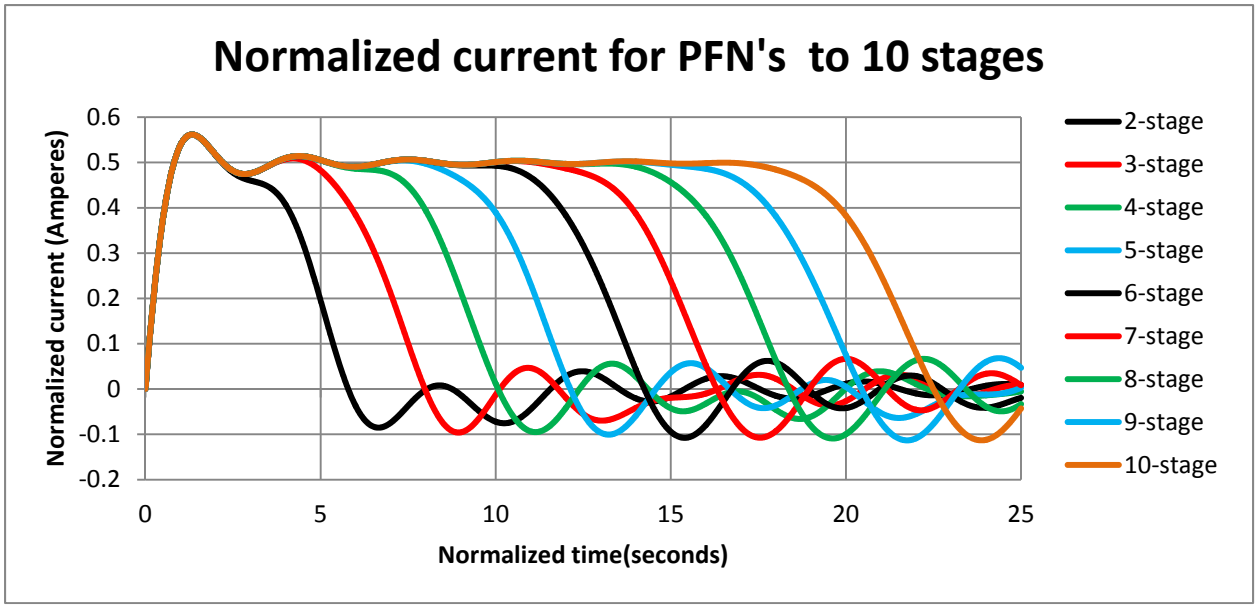
Parsing out cosine and sine terms completes the solution. Each of the four currents will be expressed as the sum of four pairs of terms, each pair being a sinusoidal exponential decay. For the fourth-order pulse forming network, the complete solution in the normalized time domain is the following:

| | Root pair #1 | | Root pair #2 | | Root pair #2 | | Root pair #4 | |
|---|---|---|---|---|---|---|---|---|
| Real exponent = | −0.272222 | | −0.147789 | | −0.064035 | | −0.015954 | |
| Frequency = | 0.278653 | | 0.913548 | | 1.484789 | | 1.866410 | |
| | Coefficient of | | Coefficient of | | Coefficient of | | Coefficient of | |
| | cosine | sine | cosine | sine | cosine | sine | cosine | sine |
| $I_4(t) = \sum$ | 0.11216 | 0.60246 | −0.18187 | −0.33426 | 0.09507 | 0.14513 | 0.02536 | 0.03679 |
| $I_3(t) = \sum$ | 0.02037 | 0.61734 | 0.05620 | −0.11171 | −0.14172 | 0.15615 | 0.06516 | 0.08985 |
| $I_2(t) = \sum$ | −0.16516 | 0.63312 | 0.27876 | 0.21681 | 0.03696 | 0.14077 | 0.07664 | 0.09259 |
| $I_1(t) = \sum$ | −0.44615 | 0.62161 | 0.21622 | 0.44439 | 0.17590 | 0.17735 | 0.05403 | 0.04293 |

(54)

I have attached hereto as Appendix "E" the solutions in the normalized time domain for all orders of pulse forming networks up to the 10th order. I have also attached, as Appendix "F", a listing of the Visual Basic program used to derive all of the numerical results above. The version of VB used is Visual Basic 2010 Express from Microsoft.

The following graph shows the current through the load resistor, that is, sum of the currents in all loops, for all orders of pulse-forming networks up to the 10th order. Both the horizontal time axis and the vertical current axis are normalized. I will explain how to convert the axes for a particular set of components or desired pulse duration after we look at the graph.




Adding an additional stage has the effect of extending the duration of the pulse by a set amount. It does not affect the magnitude of the current.

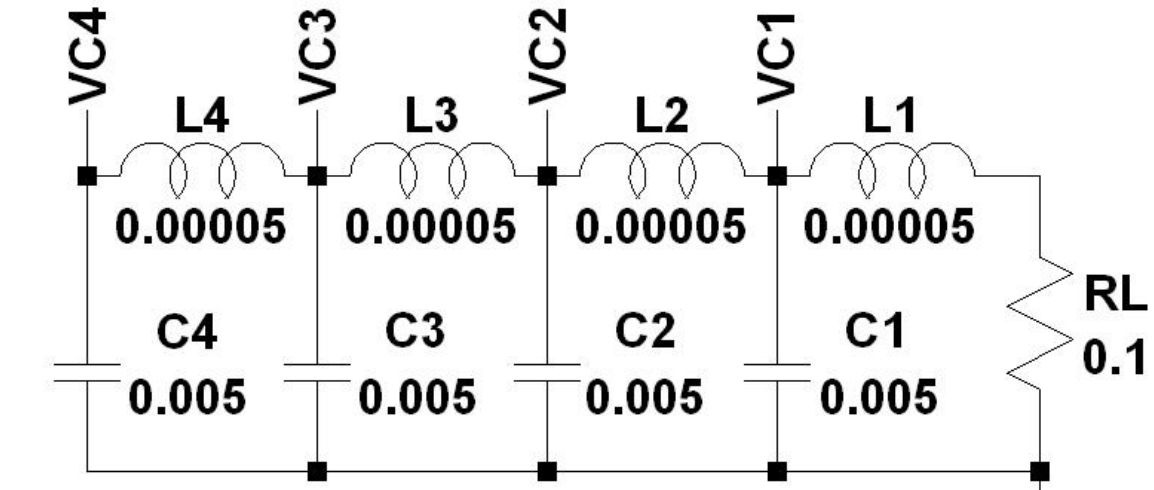
Let us look at a specific example, using the four-stage network. Suppose we want a pulse duration of say, four milliseconds. Since the pulse duration measure, $T$, approximates twice the pulse duration, we shall set $T = 0.002$ seconds. Let us use the same load resistor as before: $R = 0.1\ \Omega$. Since $N = 4$ is the number of stages, the value of the capacitors and inductors are given by Equation (24) as:

$$\left.\begin{aligned} C &= \frac{T}{NR} = \frac{0.002}{4 \times 0.1} = 0.005 \text{ F} \\ L &= \frac{RT}{N} = \frac{0.1 \times 0.002}{4} = 50 \text{ mH} \end{aligned}\right\} \quad (24)$$

Lastly, let us assume that we want to send a current of 75 Amperes through the load resistor. Since all of the coefficients in the terms whose sums are the currents share the common factor $(T/N)\,V_C(0)/L$, and since the square wave pulse has a normalized magnitude of one-half, we can write:

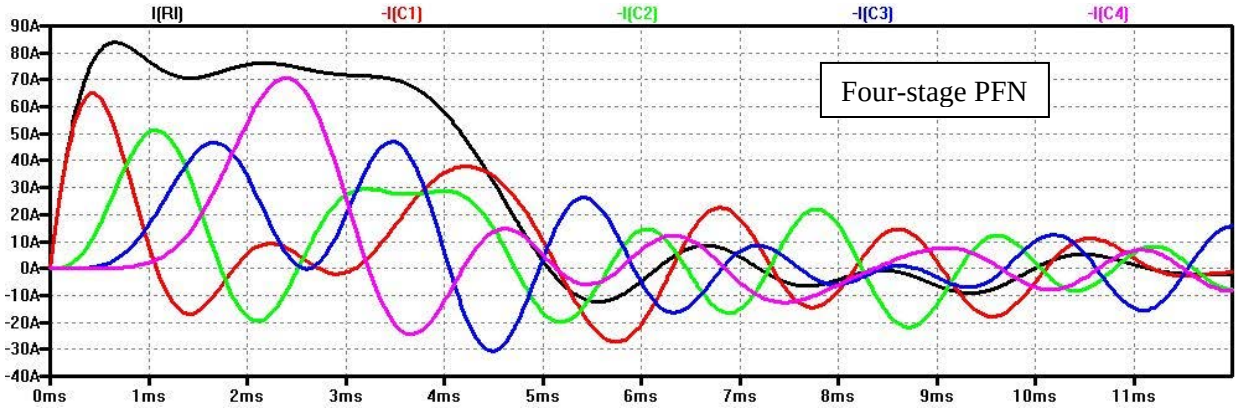
$$\begin{aligned} \frac{T}{N}\frac{V_C(0)}{L} &= 2 \times 75 \\ \rightarrow \quad V_C(0) &= 2 \times 75 \times \frac{NL}{T} \\ &= 2 \times 75 \times \frac{4 \times 0.00005}{0.002} \\ &= 15 \text{ Volts} \end{aligned} \quad (55)$$

This circuit was simulated using the following SPICE schematic.




```
.ic V(VC1)=15
.ic V(VC2)=15
.ic V(VC3)=15  .tran 0 0.012 0 0.000001 startup uic
.ic V(VC4)=15
```

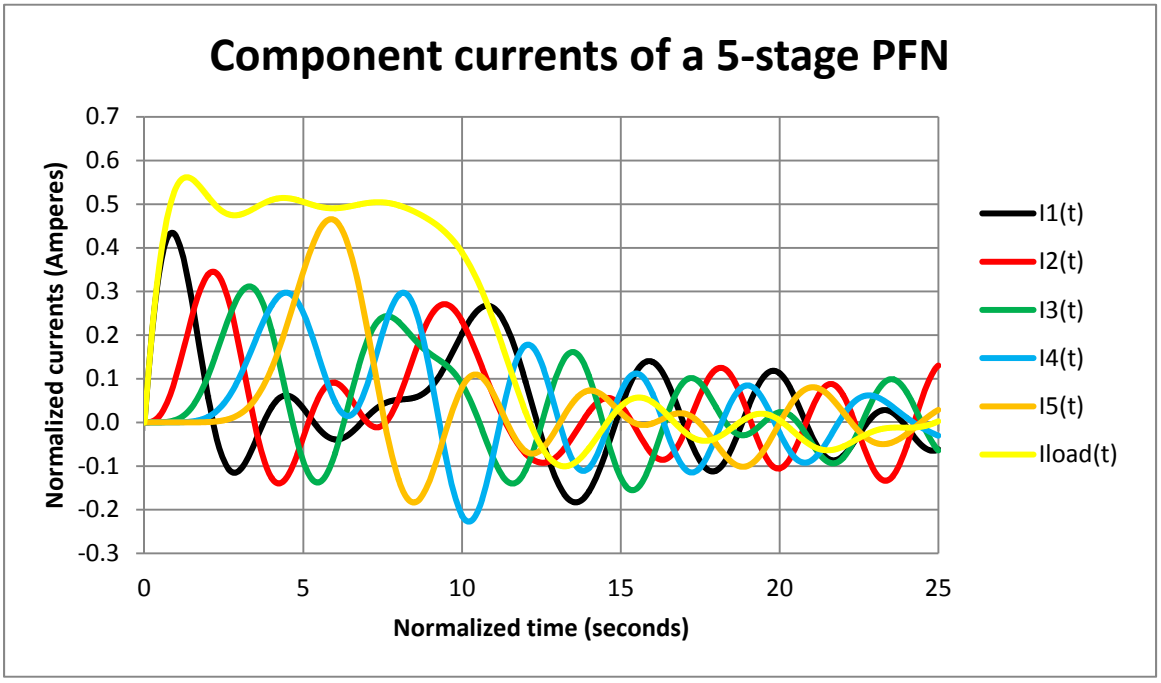
The results from the simulation are shown in the following graph. The black curve is the curve flowing through the load resistor. It approximates a square wave with duration . This black curve should be compared with the four-stage curve in the set of normalized graphs above. For the sake of comparison, I have also shown in the following graph the four individual loop currents.

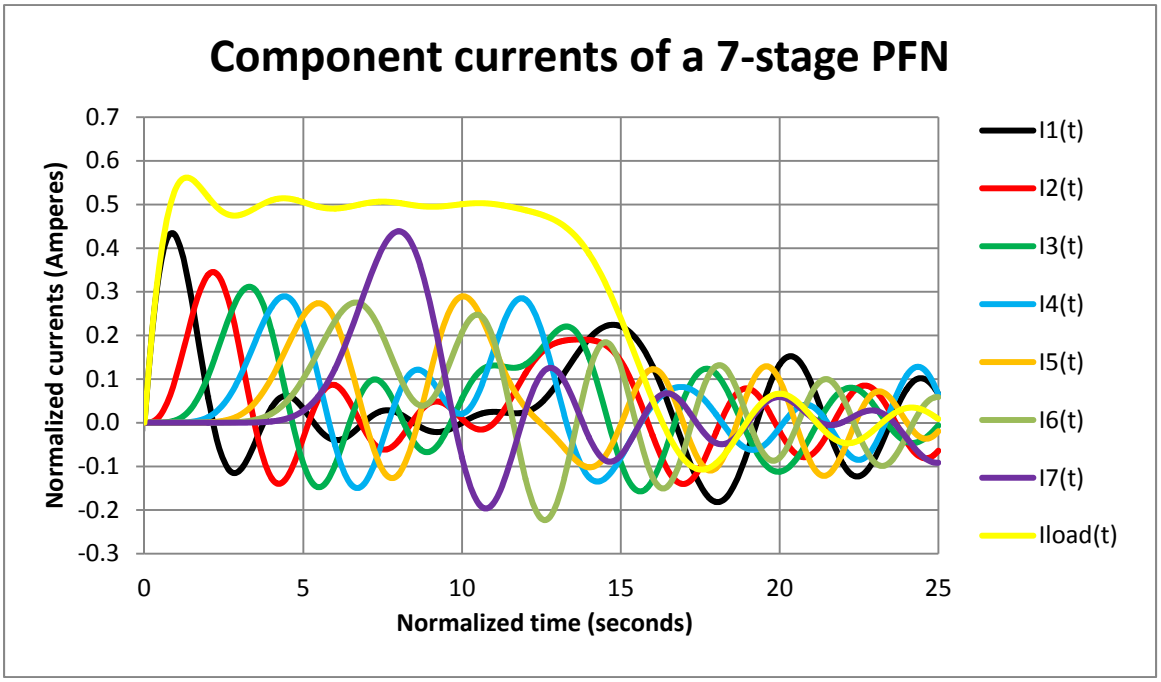



For a four-stage network, the four capacitor "fire" in order. Even so, the waveforms of the individual currents are not as regular as one might have expected. (Alert readers may notice that the four individual loop currents in this graph are preceded by minus signs in the header above the graph. This graph was produced by the SPICE simulator. SPICE defines the currents associated with a component like a capacitor as being positive when it flows into the component. This is the opposite of the convention we used when labeling the currents in the schematic shown at the outset of this section.)

In the four-stage network, the last stage (the waveform shown in purple in the graph) seems to generate a current pulse which is disproportionately larger than the current pulses generated by the other stages. This behavior is shared by the other orders of networks, too. For example, the following graphs show the

component currents of a five-stage and a seven-stage network, respectively. The last stage always generates a larger pulse.




In this paper so far, we have considered the capacitors and inductors to be ideal. We have also assumed that the load is entirely resistive. Real circuits are not so ideal. In Part II of this paper, we will look at the effects of such practical matters on pulse forming networks.

One last matter – I have not said why these networks are called “Rayleigh” pulse forming networks, rather than something else. In 1893, J.J. Thomson proposed that a three-dimension structure could be used as a guide for electromagnetic waves. We now call such a structure a “waveguide”. The following year, Oliver Lodge tested the idea. Then, in 1897, the great physicist Lord Rayleigh completed the first mathematical analysis of the progression of electromagnetic waves in a metal cylinder. He modeled the waves and cylinder as currents and voltages propagating through an electrical circuit. The circuit he used was the pulse forming network we examined in this paper. He used this kind of network, not as a store of energy to be released in a square wave, but as a means to transmit energy. Indeed, electrical transmission lines are still modeled using cascading capacitors and inductors. We have used Rayleigh’s original configuration, in which the inductors are all in series with the load resistor and the capacitors define the loops, or stages. Readers should be aware that other configurations can and are used. For example, the inductors can be placed in series with the capacitors, with a capacitor and inductor in the individual loops. Or, the capacitors and inductors can be switched.

Jim Hawley
August 2012

An e-mail setting out errors and omissions would be appreciated.

## Appendix "A"

**Expansions for the currents in each loop $A_i(t)$ in terms of the current in the final stage $A_N(t)$**

For example, $A_{N-4} = (x^8 + 7x^6 + 15x^4 + 10x^2 + 1)A_N$

| Power | $x^0$ | $x^2$ | $x^4$ | $x^6$ | $x^8$ | $x^{10}$ | $x^{12}$ | $x^{14}$ | $x^{16}$ | $x^{18}$ | $x^{20}$ | $x^{22}$ | $x^{24}$ | $x^{26}$ | $x^{28}$ | $x^{30}$ | $x^{32}$ | $x^{34}$ | $x^{36}$ | $x^{36}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Stage | | | | | | | | | | | | | | | | | | | | |
| 20 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 19 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 18 | 1 | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 17 | 1 | 6 | 5 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 16 | 1 | 10 | 15 | 7 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 15 | 1 | 15 | 35 | 28 | 9 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | 1 | 21 | 70 | 84 | 45 | 11 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13 | 1 | 28 | 126 | 210 | 165 | 66 | 13 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | 1 | 36 | 210 | 462 | 495 | 286 | 91 | 15 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | 1 | 45 | 330 | 924 | 1,287 | 1,001 | 455 | 120 | 17 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 1 | 55 | 495 | 1,716 | 3,003 | 3,003 | 1,820 | 680 | 153 | 19 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 1 | 66 | 715 | 3,003 | 6,435 | 8,008 | 6,188 | 3,060 | 969 | 190 | 21 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 1 | 78 | 1,001 | 5,005 | 12,870 | 19,448 | 18,564 | 11,628 | 4,845 | 1,330 | 231 | 23 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 1 | 91 | 1,365 | 8,008 | 24,310 | 43,758 | 50,388 | 38,760 | 20,349 | 7,315 | 1,771 | 276 | 25 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 1 | 105 | 1,820 | 12,376 | 43,758 | 92,378 | 125,970 | 116,280 | 74,613 | 33,649 | 10,626 | 2,300 | 325 | 27 | 1 | 0 | 0 | 0 | 0 | 0 |
| 5 | 1 | 120 | 2,380 | 18,564 | 75,582 | 184,756 | 293,930 | 319,770 | 245,157 | 134,596 | 53,130 | 14,950 | 2,925 | 378 | 29 | 1 | 0 | 0 | 0 | 0 |
| 4 | 1 | 136 | 3,060 | 27,132 | 125,970 | 352,716 | 646,646 | 817,190 | 735,471 | 480,700 | 230,230 | 80,730 | 20,475 | 3,654 | 435 | 31 | 1 | 0 | 0 | 0 |
| 3 | 1 | 153 | 3,876 | 38,760 | 203,490 | 646,646 | 1,352,078 | 1,961,256 | 2,042,975 | 1,562,275 | 888,030 | 376,740 | 118,755 | 27,405 | 4,495 | 496 | 33 | 1 | 0 | 0 |
| 2 | 1 | 171 | 4,845 | 54,264 | 319,770 | 1,144,066 | 2,704,156 | 4,457,400 | 5,311,735 | 4,686,825 | 3,108,105 | 1,560,780 | 593,775 | 169,911 | 35,960 | 5,456 | 561 | 35 | 1 | 0 |
| 1 | 1 | 190 | 5,985 | 74,613 | 490,314 | 1,961,256 | 5,200,300 | 9,657,700 | 13,037,895 | 13,123,110 | 10,015,005 | 5,852,925 | 2,629,575 | 906,192 | 237,336 | 46,376 | 6,545 | 630 | 37 | 1 |

## Appendix "B" (page 1 of 4)

### Characteristic equations for pulse forming networks

As an example, for a fourth-order network:

$$(x^8 + x^7 + 7x^6 + 8x^5 + 15x^4 + 10x^3 + 10x^2 + 4x + 1)A_4 = 0$$

| Power | $x^0$ | $x^1$ | $x^2$ | $x^3$ | $x^4$ | $x^5$ | $x^6$ | $x^7$ | $x^8$ | $x^9$ | $x^{10}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Order | | | | | | | | | | | |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | 2 | 3 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 1 | 3 | 6 | 4 | 5 | 1 | 1 | 0 | 0 | 0 | 0 |
| 4 | 1 | 4 | 10 | 10 | 15 | 6 | 7 | 1 | 1 | 0 | 0 |
| 5 | 1 | 5 | 15 | 20 | 35 | 21 | 28 | 8 | 9 | 1 | 1 |
| 6 | 1 | 6 | 21 | 35 | 70 | 56 | 84 | 36 | 45 | 10 | 11 |
| 7 | 1 | 7 | 28 | 56 | 126 | 126 | 210 | 120 | 165 | 55 | 66 |
| 8 | 1 | 8 | 36 | 84 | 210 | 252 | 462 | 330 | 495 | 220 | 286 |
| 9 | 1 | 9 | 45 | 120 | 330 | 462 | 924 | 792 | 1,287 | 715 | 1,001 |
| 10 | 1 | 10 | 55 | 165 | 495 | 792 | 1,716 | 1,716 | 3,003 | 2,002 | 3,003 |
| 11 | 1 | 11 | 66 | 220 | 715 | 1,287 | 3,003 | 3,432 | 6,435 | 5,005 | 8,008 |
| 12 | 1 | 12 | 78 | 286 | 1,001 | 2,002 | 5,005 | 6,435 | 12,870 | 11,440 | 19,448 |
| 13 | 1 | 13 | 91 | 364 | 1,365 | 3,003 | 8,008 | 11,440 | 24,310 | 24,310 | 43,758 |
| 14 | 1 | 14 | 105 | 455 | 1,820 | 4,368 | 12,376 | 19,448 | 43,758 | 48,620 | 92,378 |
| 15 | 1 | 15 | 120 | 560 | 2,380 | 6,188 | 18,564 | 31,824 | 75,582 | 92,378 | 184,756 |
| 16 | 1 | 16 | 136 | 680 | 3,060 | 8,568 | 27,132 | 50,388 | 125,970 | 167,960 | 352,716 |
| 17 | 1 | 17 | 153 | 816 | 3,876 | 11,628 | 38,760 | 77,520 | 203,490 | 293,930 | 646,646 |
| 18 | 1 | 18 | 171 | 969 | 4,845 | 15,504 | 54,264 | 116,280 | 319,770 | 497,420 | 1,144,066 |
| 19 | 1 | 19 | 190 | 1,140 | 5,985 | 20,349 | 74,613 | 170,544 | 490,314 | 817,190 | 1,961,256 |
| 20 | 1 | 20 | 210 | 1,330 | 7,315 | 26,334 | 100,947 | 245,157 | 735,471 | 1,307,504 | 3,268,760 |

As an example, for a fourth-order network:

$$(x^8 + x^7 + 7x^6 + 8x^5 + 15x^4 + 10x^3 + 10x^2 + 4x + 1)A_4 = 0$$

| Power | $x^{11}$ | $x^{12}$ | $x^{13}$ | $x^{14}$ | $x^{15}$ | $x^{16}$ | $x^{17}$ | $x^{18}$ | $x^{19}$ | $x^{20}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| Order | | | | | | | | | | |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 12 | 13 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 78 | 91 | 14 | 15 | 1 | 1 | 0 | 0 | 0 | 0 |
| 9 | 364 | 455 | 105 | 120 | 16 | 17 | 1 | 1 | 0 | 0 |
| 10 | 1,365 | 1,820 | 560 | 680 | 136 | 153 | 18 | 19 | 1 | 1 |
| 11 | 4,368 | 6,188 | 2,380 | 3,060 | 816 | 969 | 171 | 190 | 20 | 21 |
| 12 | 12,376 | 18,564 | 8,568 | 11,628 | 3,876 | 4,845 | 1,140 | 1,330 | 210 | 231 |
| 13 | 31,824 | 50,388 | 27,132 | 38,760 | 15,504 | 20,349 | 5,985 | 7,315 | 1,540 | 1,771 |
| 14 | 75,582 | 125,970 | 77,520 | 116,280 | 54,264 | 74,613 | 26,334 | 33,649 | 8,855 | 10,626 |
| 15 | 167,960 | 293,930 | 203,490 | 319,770 | 170,544 | 245,157 | 100,947 | 134,596 | 42,504 | 53,130 |
| 16 | 352,716 | 646,646 | 497,420 | 817,190 | 490,314 | 735,471 | 346,104 | 480,700 | 177,100 | 230,230 |
| 17 | 705,432 | 1,352,078 | 1,144,066 | 1,961,256 | 1,307,504 | 2,042,975 | 1,081,575 | 1,562,275 | 657,800 | 888,030 |
| 18 | 1,352,078 | 2,704,156 | 2,496,144 | 4,457,400 | 3,268,760 | 5,311,735 | 3,124,550 | 4,686,825 | 2,220,075 | 3,108,105 |
| 19 | 2,496,144 | 5,200,300 | 5,200,300 | 9,657,700 | 7,726,160 | 13,037,895 | 8,436,285 | 13,123,110 | 6,906,900 | 10,015,005 |
| 20 | 4,457,400 | 9,657,700 | 10,400,600 | 20,058,300 | 17,383,860 | 30,421,755 | 21,474,180 | 34,597,290 | 20,030,010 | 30,045,015 |

**Characteristic equations for pulse forming networks**

As an example, for a fourth-order network:

$$(x^8 + x^7 + 7x^6 + 8x^5 + 15x^4 + 10x^3 + 10x^2 + 4x + 1)A_4 = 0$$

| Power Order | $x^{21}$ | $x^{22}$ | $x^{23}$ | $x^{24}$ | $x^{25}$ | $x^{26}$ | $x^{27}$ | $x^{28}$ | $x^{29}$ | $x^{30}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | 22 | 23 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13 | 253 | 276 | 24 | 25 | 1 | 1 | 0 | 0 | 0 | 0 |
| 14 | 2,024 | 2,300 | 300 | 325 | 26 | 27 | 1 | 1 | 0 | 0 |
| 15 | 12,650 | 14,950 | 2,600 | 2,925 | 351 | 378 | 28 | 29 | 1 | 1 |
| 16 | 65,780 | 80,730 | 17,550 | 20,475 | 3,276 | 3,654 | 406 | 435 | 30 | 31 |
| 17 | 296,010 | 376,740 | 98,280 | 118,755 | 23,751 | 27,405 | 4,060 | 4,495 | 465 | 496 |
| 18 | 1,184,040 | 1,560,780 | 475,020 | 593,775 | 142,506 | 169,911 | 31,465 | 35,960 | 4,960 | 5,456 |
| 19 | 4,292,145 | 5,852,925 | 2,035,800 | 2,629,575 | 736,281 | 906,192 | 201,376 | 237,336 | 40,920 | 46,376 |
| 20 | 14,307,150 | 20,160,075 | 7,888,725 | 10,518,300 | 3,365,856 | 4,272,048 | 1,107,568 | 1,344,904 | 278,256 | 324,632 |

**Characteristic equations for pulse forming networks**

As an example, for a fourth-order network:

$$(x^8 + x^7 + 7x^6 + 8x^5 + 15x^4 + 10x^3 + 10x^2 + 4x + 1)A_4 = 0$$

| Power Order | $x^{31}$ | $x^{32}$ | $x^{33}$ | $x^{34}$ | $x^{35}$ | $x^{36}$ | $x^{37}$ | $x^{38}$ | $x^{39}$ | $x^{40}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 15 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 16 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 17 | 32 | 33 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 18 | 528 | 561 | 34 | 35 | 1 | 1 | 0 | 0 | 0 | 0 |
| 19 | 5,984 | 6,545 | 595 | 630 | 36 | 37 | 1 | 1 | 0 | 0 |
| 20 | 52,360 | 58,905 | 7,140 | 7,770 | 666 | 703 | 38 | 39 | 1 | 1 |

## Appendix "C"

## Roots of the characteristic equations

| 2-stage network | | | 3-stage network | | | 4-stage network | | |
|---|---|---|---|---|---|---|---|---|
| | Re part | Im part | | Re part | Im part | | Re part | Im part |
| Root 1 | -0.39512338 | 0.50684390 | Root 1 | -0.32160832 | 0.35907902 | Root 1 | -0.27222242 | 0.27865255 |
| Root 2 | -0.39512338 | -0.50684390 | Root 2 | -0.32160832 | -0.35907902 | Root 2 | -0.27222242 | -0.27865255 |
| Root 3 | -0.10487662 | 1.55249182 | Root 3 | -0.14292369 | 1.15951566 | Root 3 | -0.14778932 | 0.91354784 |
| Root 4 | -0.10487662 | -1.55249182 | Root 4 | -0.14292369 | -1.15951566 | Root 4 | -0.14778932 | -0.91354784 |
| | | | Root 5 | -0.03546799 | 1.77530357 | Root 5 | -0.06403473 | 1.48478891 |
| | | | Root 6 | -0.03546799 | -1.77530357 | Root 6 | -0.06403473 | -1.48478891 |
| | | | | | | Root 7 | -0.01595353 | 1.86641011 |
| | | | | | | Root 8 | -0.01595353 | -1.86641011 |
| **5-stage network** | | | **6-stage network** | | | **7-stage network** | | |
| | Re part | Im part | | Re part | Im part | | Re part | Im part |
| Root 1 | -0.23700180 | 0.22800318 | Root 1 | -0.21057366 | 0.19313105 | Root 1 | -0.18995523 | 0.16763201 |
| Root 2 | -0.23700180 | -0.22800318 | Root 2 | -0.21057366 | -0.19313105 | Root 2 | -0.18995523 | -0.16763201 |
| Root 3 | -0.03399643 | 1.65519310 | Root 3 | -0.04561456 | 1.46125015 | Root 3 | -0.05235863 | 1.29832262 |
| Root 4 | -0.03399643 | -1.65519310 | Root 4 | -0.04561456 | -1.46125015 | Root 4 | -0.05235863 | -1.29832262 |
| Root 5 | -0.14315963 | 0.75090351 | Root 5 | -0.13597034 | 0.63653835 | Root 5 | -0.02916481 | 1.59391269 |
| Root 6 | -0.14315963 | -0.75090351 | Root 6 | -0.13597034 | -0.63653835 | Root 6 | -0.02916481 | -1.59391269 |
| Root 7 | -0.07735643 | 1.25439804 | Root 7 | -0.00503668 | 1.93747412 | Root 7 | -0.12842337 | 0.55207639 |
| Root 8 | -0.07735643 | -1.25439804 | Root 8 | -0.00503668 | -1.93747412 | Root 8 | -0.12842337 | -0.55207639 |
| Root 9 | -0.00848571 | 1.91176780 | Root 9 | -0.08264187 | 1.07907395 | Root 9 | -0.00322956 | 1.95341044 |
| Root 10 | -0.00848571 | -1.91176780 | Root 10 | -0.08264187 | -1.07907395 | Root 10 | -0.00322956 | -1.95341044 |
| | | | Root 11 | -0.02016289 | 1.75391911 | Root 11 | -0.08394437 | 0.94400876 |
| | | | Root 12 | -0.02016289 | -1.75391911 | Root 12 | -0.08394437 | -0.94400876 |
| | | | | | | Root 13 | -0.01292403 | 1.81585756 |
| | | | | | | Root 14 | -0.01292403 | -1.81585756 |
| **8-stage network** | | | **9-stage network** | | | **10-stage network** | | |
| | Re part | Im part | | Re part | Im part | | Re part | Im part |
| Root 1 | -0.17337820 | 0.14815989 | Root 1 | -0.15973043 | 0.13279477 | Root 1 | -0.14827732 | 0.12035553 |
| Root 2 | -0.17337820 | -0.14815989 | Root 2 | -0.15973043 | -0.13279477 | Root 2 | -0.14827732 | -0.12035553 |
| Root 3 | -0.05602035 | 1.16372067 | Root 3 | -0.11464328 | 0.43602509 | Root 3 | -0.10865540 | 0.39451177 |
| Root 4 | -0.05602035 | -1.16372067 | Root 4 | -0.11464328 | -0.43602509 | Root 4 | -0.10865540 | -0.39451177 |
| Root 5 | -0.03535442 | 1.44942347 | Root 5 | -0.01402582 | 1.74664331 | Root 5 | -0.01836333 | 1.63661490 |
| Root 6 | -0.03535442 | -1.44942347 | Root 6 | -0.01402582 | -1.74664331 | Root 6 | -0.01836333 | -1.63661490 |
| Root 7 | -0.12124269 | 0.48727002 | Root 7 | -0.08158951 | 0.75228604 | Root 7 | -0.01030662 | 1.79273270 |
| Root 8 | -0.12124269 | -0.48727002 | Root 8 | -0.08158951 | -0.75228604 | Root 8 | -0.01030662 | -1.79273270 |
| Root 9 | -0.00877436 | 1.85715197 | Root 9 | -0.05778834 | 1.05218903 | Root 9 | -0.00457747 | 1.90696484 |
| Root 10 | -0.00877436 | -1.85715197 | Root 10 | -0.05778834 | -1.05218903 | Root 10 | -0.00457747 | -1.90696484 |
| Root 11 | -0.08326055 | 0.83771917 | Root 11 | -0.03941929 | 1.32341676 | Root 11 | -0.04199225 | 1.21456130 |
| Root 12 | -0.08326055 | -0.83771917 | Root 12 | -0.03941929 | -1.32341676 | Root 12 | -0.04199225 | -1.21456130 |
| Root 13 | -0.01977643 | 1.68350148 | Root 13 | -0.00155648 | 1.97129545 | Root 13 | -0.02884373 | 1.44233129 |
| Root 14 | -0.01977643 | -1.68350148 | Root 14 | -0.00155648 | -1.97129545 | Root 14 | -0.02884373 | -1.44233129 |
| Root 15 | -0.00219300 | 1.96395813 | Root 15 | -0.02501988 | 1.55724953 | Root 15 | -0.07944791 | 0.68230138 |
| Root 16 | -0.00219300 | -1.96395813 | Root 16 | -0.02501988 | -1.55724953 | Root 16 | -0.07944791 | -0.68230138 |
| | | | Root 17 | -0.00622697 | 1.88601592 | Root 17 | -0.05839171 | 0.95893832 |
| | | | Root 18 | -0.00622697 | -1.88601592 | Root 18 | -0.05839171 | -0.95893832 |
| | | | | | | Root 19 | -0.00114424 | 1.97660299 |
| | | | | | | Root 20 | -0.00114424 | -1.97660299 |

## Appendix "D"

## Finding roots of polynomials with real coefficients

The polynomials of interest in this paper have real coefficients and involve only even powers of the independent variable. The typical form is:

$$P(x) = a_n x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_2 x^2 + a_1 x + a_0 \quad (A1)$$

If the coefficients are real and $n$ is even, the polynomial $P(x)$ can be expressed as the product of $\frac{1}{2}n$ quadratic expressions. The technique described here will work even if $n$ is odd and greater than 3, in which case there will be $\frac{1}{2}(n-1)$ quadratic expressions.

If $P(x)$ can be factored into at least one quadratic expression, then we can divide it by <u>any</u> quadratic expression $x^2 + px + q$ and write:

$$P(x) = (x^2 + px + q)(b_{n-2}x^{n-2} + b_{n-3}x^{n-3} + \cdots + b_2 x^2 + b_1 x + b_0) + Rx + S \quad (A2)$$

where $Rx + S$ is the remainder from the division. Of course, we are going to be looking for the pair of $p$ and $q$ which result in the remainder being zero, in which case we will have found the factoring quadratic. The expression for $P(x)$ in Equation $(A2)$ can be expanded into its various powers of $x$ as follows:

$$P(x) = \begin{bmatrix} b_{n-2}x^n + \cdots \\ \cdots + (pb_{n-2} + b_{n-3})x^{n-1} + \cdots \\ \cdots + (qb_{n-2} + pb_{n-3} + b_{n-4})x^{n-2} + \cdots \\ \cdots + (qb_{n-3} + pb_{n-4} + b_{n-5})x^{n-3} + \cdots \\ \cdots \\ \cdots + (qb_3 + pb_2 + b_1)x^3 + \cdots \\ \cdots + (qb_2 + pb_1 + b_0)x^2 + \cdots \\ \cdots + (qb_1 + pb_0 + R)x + \cdots \\ \cdots + (qb_0 + S) \end{bmatrix} \quad (A3)$$

Since the expressions for $P(x)$ in Equations $(A1)$ and $(A3)$ must be the same for any and all values of $x$, we can set equal the coefficients of each order of $x$. This gives $n + 1$ simultaneous equations.

$$\left.\begin{array}{l} a_n = b_{n-2} \\ a_{n-1} = pb_{n-2} + b_{n-3} \\ a_{n-2} = qb_{n-2} + pb_{n-3} + b_{n-4} \\ a_{n-3} = qb_{n-3} + pb_{n-4} + b_{n-5} \\ \cdots \\ a_3 = qb_3 + pb_2 + b_1 \\ a_2 = qb_2 + pb_1 + b_0 \\ a_1 = qb_1 + pb_0 + R \\ a_0 = qb_0 + S \end{array}\right\} \quad (A4)$$

If we define $b_n = 0$ and $b_{n-1} = 0$, then we can write the $b$ coefficients as follows:

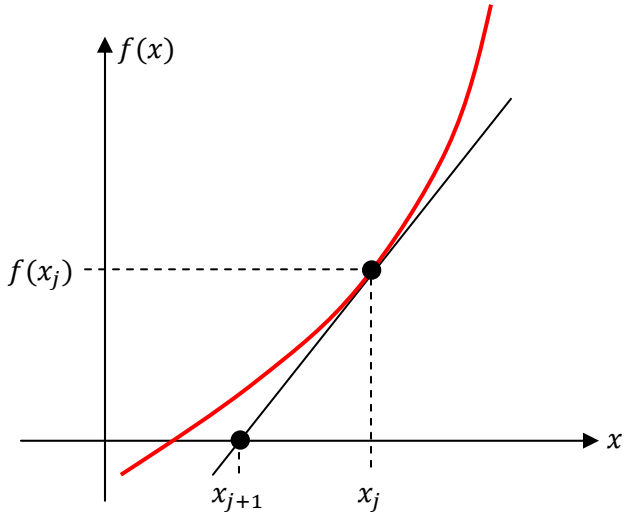
$$\left.\begin{aligned} b_n &= 0 \\ b_{n-1} &= 0 \\ b_k &= a_{k+2} - qb_{k+2} - pb_{k+1} \quad \text{for } k = n-2, n-1, \cdots, 2, 1, 0 \end{aligned}\right\} \quad (A5)$$

Each coefficient $b_k$ is a linear combination of the next two higher $b_k$'s. In addition, one can think of each $b_k$ as being a function of $p$ and $q$. Similarly, $R$ and $S$ can be considered to be the following functions of $p$ and $q$:

$$\left.\begin{aligned} R &= a_1 - qb_1 - pb_0 \\ S &= a_0 - qb_0 \end{aligned}\right\} \quad (A6)$$

The solution for $p$ and $q$ which gives $R(p,q) = 0$ and $S(p,q) = 0$ defines the quadratic expression which can be factored out of $P(x)$.

It was Bairstow who suggested that the two functions for $R$ and $S$ in the two independent variables $p$ and $q$ could be solved using Newton's method of successive approximations. The principle behind Newton's method of successive approximations for a function $f$ which depends on a single independent variable $f = f(x)$ is easy to understand (and for me to depict).




The function itself is shown in red in the figure. Assume that we have already found an approximation for the root, which approximation we can call $x_j$. The slope of the function at $x = x_j$ is shown as the black line. If we know the slope of the function at this value of $x$, then the next, and better, approximation, will be at the value $x_{j+1}$ where the tangent line intersects the $x$-axis. Mathematically, the progression to the next approximation for the root is given setting equal the following two expressions for the slope of the function at the abscissa $x_j$:

$$\left.\begin{aligned} slope &= \left.\frac{df}{dx}\right|_{x=x_j} \\ slope &= \frac{f(x_j)}{x_j - x_{j+1}} \end{aligned}\right\} \quad (A7)$$

A function $f$ which depends on two independent variables $f = f(p,q)$ can be approached in the same way (but is much tougher for me to depict in a meaningful way. One needs to consider the slope of $f$ with respect to the $p$-variable, and to get the next, and better, approximation for the root along the $p$-axis.

At the same time, one needs to consider the slope of $f$ with respect to the $q$-variable, and to get the next, and better, approximation for the root along the $q$-axis. In this case, though, function $f$ has two, independent derivatives. They are called "partial derivatives". The partial derivative of $f$ with respect to the $p$-variable measures how quickly $f$ changes as $p$ is changed, while leaving $q$ constant. Similarly, the partial derivative of $f$ with respect to the $q$-variable measures how quickly $f$ changes as $q$is changed, while leaving $p$ constant. The differential in $f$ can be written as a linear combination of the differential in $p$ and the differential in $q$, thus:

$$df = \frac{\partial f}{\partial p}dp + \frac{\partial f}{\partial q}dq \quad (A8)$$

In the same way as for the univariate case, the progression from approximations $p_j$ and $q_j$ for the roots to the next, and better, pair of approximations $p_{j+1}$ and $q_{j+1}$, is given by setting equal the expressions for two different slopes, as follows:

$$\left.\begin{aligned} \text{slope in } p-\text{direction} &= \left.\frac{\partial f}{\partial p}\right|_{p=p_j,\ q=q_j} \\ \text{slope in } p-\text{direction} &= \frac{f(p_j,q_j)}{p_j - p_{j+1}} \\ \text{slope in } q-\text{direction} &= \left.\frac{\partial f}{\partial q}\right|_{p=p_j,\ q=q_j} \\ \text{slope in } q-\text{direction} &= \frac{f(p_j,q_j)}{q_j - q_{j+1}} \end{aligned}\right\} \quad (A9)$$

Let us return to our situation, where we are looking for the roots $p$ and $q$ at which the two functions $R$ and $S$ are equal to zero. One could follow in the footsteps of Bairstow and derive closed-form expressions for the partial derivatives of $R$ and $S$ in terms of $p$ and $q$ and, then, in terms of the $b_k$ coefficients. This is involved but does give an efficient algorithm. Or, one could follow in my footsteps and use a quick trick. The trick is only about one-third as efficient as Bairstow's algorithm, but is much quicker to debug numerically.

The trick is this: instead of using a closed-form solution for the partial derivatives, calculate them directly as differences. If the current approximation for $p$ and $q$ are $p_j$ and $q_j$, respectively, then find the next pair of approximations by:

Step #1: Evaluate the function at the known point. For example, calculate $R(p_j, q_j)$.
Step #2: Consider a point very close to the known point, with the same $q$, but a slightly greater $p$. Calculate the function at this point $R(p_j + \Delta p, q_j)$.
Step #3: Consider a second point very close to the known point, with the same $p$, but a slightly greater $q$. Calculate the function at this point $R(p_j, q_j + \Delta q)$.
Step #4: The partial derivatives at the known point can be approximated as:

$$\left.\begin{aligned} \left.\frac{\partial f}{\partial p}\right|_{p=p_j,\ q=q_j} &\cong \frac{R(p_j+\Delta p, q_j) - R(p_j,q_j)}{\Delta p} \\ \left.\frac{\partial f}{\partial q}\right|_{p=p_j,\ q=q_j} &\cong \frac{R(p_j, q_j+\Delta q) - R(p_j,q_j)}{\Delta q} \end{aligned}\right\} \quad (A10)$$

In the limit as $\Delta p$ and $\Delta q$ are made very small, the ratio of differences in Equation $(A10)$ become the derivatives themselves. One must, however, evaluate the function three times for every step in the progression. This is not a problem if one only needs to calculate a couple of dozen roots, as in this paper.

Just to be clear, $R$ and $S$ are calculated for a given pair of $p$ and $q$ as follows. All of the $b_k$ coefficients are calculated recursively using Equation $(A5)$, starting with $b_{n-2}$ and ending with $b_0$. Once $b_1$ and $b_0$ have been computed, they can be combined with $p$ and $q$ using Equation $(A6)$ to calculate $R$ and $S$.

Additional note #1:

Newton's method of successive approximations needs an initial guess for $p$ and $q$. In the numerical procedure whose listing is attached as Appendix "B", the initial guesses I made were:

$$p_{guess} = \frac{a_1}{2\sqrt{a_0}} \quad (A11a)$$

$$q_{guess} = \sqrt{a_0} \quad (A11b)$$

Let me explain why these guesses make sense. Let us consider $q_{guess}$ first. The second of Equations $(A6)$ has $S = a_0 - qb_0$. We know nothing about $S$ except that we want it to be zero. Setting it to zero gives $a_0 = qb_0$. We know nothing about $b_0$ so, as a first guess, let us set it equal to $q$, so the two factors on the right-hand side of this expression carry equal weight. This results in Equation $(A11b)$. Equation $(A11a)$ is derived from the first of Equations $(A6)$ in a similar way. Firstly, set $R = 0$ so that $a_1 = qb_1 + pb_0$. Secondly, since we do not know the two terms on the right-hand side of this expression, let us assume that they are equal. Thirdly, use $b_0 = \sqrt{a_0}$ from the first guess. The result is Equation $(A11a)$.

Additional note #2:

For functions which vary quickly with respect to their independent variables, Newton's method of successive approximations can be unstable. What can happen, particularly with high-order polynomials, is that the intersection between the tangent to the function at an abscissa near one root can be far away, in the neighbourhood of a completely different root. In extreme cases, the progression can find itself in a loop, where the target points of intersection alternate between different roots.

This potential difficulty can easily be overcome by limiting the amount by which either independent variable is permitted to change during one step of the progression. Instead of moving the full distance from an approximation $p_j$, say, to the target intersection which defines the next approximation $p_{j+1}$, only move a certain percentage of the distance. In the numerical procedure whose listing is attached, the fraction used was 20%.

Additional note #3:

Newton's method of successive approximations also needs guidance about when to stop. In the attached numerical procedure, the maximum "error" permitted for the final approximation is $1 \times 10^{-12}$. Four separate quantities (or, rather, their absolute values) need to be less than this value for the progression to be terminated. The quantities are:

- the value of $R$,
- the value of $S$,

- the change in $p_{j+1}$ from its value $p_j$ in the last iteration, as a fraction of $p_j$ and
- the change in $q_{j+1}$ from its value $q_j$ in the last iteration, as a fraction of $q_j$.

Additional note #4:

I promise this will be the last additional note. Extracting one quadratic expression from a polynomial or order $n$ leaves behind the quadratic expression as well as a polynomial of order $n-2$. If the purpose of the factoring is to find the roots of the original polynomial, then additional quadratic expressions must be extracted until the order of the last remaining polynomial is 2, in which case it is the last quadratic expression.

The roots of each quadratic expression can be found using the quadratic formula.

If the original polynomial had order $n$, then it can be factored into $n/2$ quadratic factors, each of which will have two roots.

**Solution for 2-stage network**

| Term-> | 1 | 2 |
|---|---|---|
| Exponent = | -0.39512338 | -0.10487662 |
| Frequency = | 0.50684390 | 1.55249182 |
| Coefficients of cosine terms | | |
| I2(t) | 0.11397100 | -0.11397100 |
| I1(t) | -0.24468104 | 0.24468104 |
| Coefficients of sine terms | | |
| I2(t) | 0.86676619 | -0.26166672 |
| I1(t) | 0.82507268 | 0.32901884 |

**Solution for 3-stage network**

| Term-> | 1 | 2 | 3 |
|---|---|---|---|
| Exponent = | -0.32160832 | -0.14292369 | -0.035467992 |
| Frequency = | 0.35907902 | 1.15951566 | 1.775303571 |
| Coefficients of cosine terms | | | |
| I3(t) | 0.11976560 | -0.17063593 | 0.050870328 |
| I2(t) | -0.04579898 | 0.16583204 | -0.120033058 |
| I1(t) | -0.37494925 | 0.26565586 | 0.109293391 |
| Coefficients of sine terms | | | |
| I3(t) | 0.70361068 | -0.33350250 | 0.084483297 |
| I2(t) | 0.71332624 | 0.05151493 | -0.175270418 |
| I1(t) | 0.69426983 | 0.42328817 | 0.102039531 |

**Solution for 4-stage network**

| Term-> | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Exponent = | -0.27222242 | -0.14778932 | -0.064034729 | -0.015953529 |
| Frequency = | 0.27865255 | 0.91354784 | 1.484788908 | 1.866410106 |
| Coefficients of cosine terms | | | | |
| I4(t) | 0.11216191 | -0.18187070 | 0.095066499 | -0.025357712 |
| I3(t) | 0.02036554 | 0.05619905 | -0.14172458 | 0.065159986 |
| I2(t) | -0.16515982 | 0.27875797 | -0.036957896 | -0.07664026 |
| I1(t) | -0.44615178 | 0.21621901 | 0.175903283 | 0.05402949 |
| Coefficients of sine terms | | | | |
| I4(t) | 0.60245533 | -0.33425892 | 0.145130606 | -0.03678991 |
| I3(t) | 0.61733750 | -0.11170702 | -0.156151477 | 0.089847795 |
| I2(t) | 0.63312262 | 0.21680746 | -0.140772397 | -0.092594845 |
| I1(t) | 0.62160854 | 0.44438814 | 0.177348243 | 0.042927803 |

**Solution for 5-stage network**

| Term-> | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Exponent = | -0.23700180 | -0.03399643 | -0.143159628 | -0.077356426 | -0.00848571 |
| Frequency = | 0.22800318 | 1.65519310 | 0.750903515 | 1.254398043 | 1.91176780 |
| Coefficients of cosine terms | | | | | |
| I5(t) | 0.10330317 | -0.05498015 | -0.178713199 | 0.116153735 | 0.01423645 |
| I4(t) | 0.04613059 | 0.10409181 | -0.012925473 | -0.098881468 | -0.03841546 |
| I3(t) | -0.06990143 | -0.03598802 | 0.199509123 | -0.144585552 | 0.05096588 |
| I2(t) | -0.24751195 | -0.07693145 | 0.293171122 | 0.079235449 | -0.04796317 |
| I1(t) | -0.48911770 | 0.10692337 | 0.163588403 | 0.188512217 | 0.03009371 |
| Coefficients of sine terms | | | | | |
| I5(t) | 0.53301132 | -0.07560206 | -0.319453329 | 0.169837859 | 0.01913283 |
| I4(t) | 0.54640608 | 0.12524729 | -0.184297722 | -0.073846024 | -0.05033162 |
| I3(t) | 0.56707275 | -0.00517948 | 0.048219201 | -0.220964092 | 0.06290874 |
| I2(t) | 0.58255773 | -0.12547234 | 0.297429761 | -0.049774155 | -0.05211515 |
| I1(t) | 0.57373069 | 0.08918390 | 0.448059712 | 0.214815627 | 0.02177439 |

**Solution for 6-stage network**

| Term-> | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Exponent = | -0.21057366 | -0.04561456 | -0.135970337 | -0.005036678 | -0.08264187 | -0.02016289 |
| Frequency = | 0.19313105 | 1.46125015 | 0.636538353 | 1.93747412 | 1.07907395 | 1.75391911 |
| Coefficients of cosine terms | | | | | | |
| I6(t) | 0.09528006 | -0.07410722 | -0.171277515 | -0.008737002 | 0.12466912 | 0.03417256 |
| I5(t) | 0.05675556 | 0.09713576 | -0.052737617 | 0.024277811 | -0.05136629 | -0.07406522 |
| I4(t) | -0.02147107 | 0.04757644 | 0.123408578 | -0.034438017 | -0.16690882 | 0.05183289 |
| I3(t) | -0.14151486 | -0.11654690 | 0.261141712 | 0.036938763 | -0.05501650 | 0.01499779 |
| I2(t) | -0.30593667 | -0.01756487 | 0.271999283 | -0.031172122 | 0.15367094 | -0.07099655 |
| I1(t) | -0.51697912 | 0.13210277 | 0.118768869 | 0.018347899 | 0.17988990 | 0.06786968 |
| Coefficients of sine terms | | | | | | |
| I6(t) | 0.48189157 | -0.09871520 | -0.302182873 | -0.01117111 | 0.17786074 | 0.04423896 |
| I5(t) | 0.49303464 | 0.10198211 | -0.214979081 | 0.030592266 | -0.00579052 | -0.08941540 |
| I4(t) | 0.51226581 | 0.09808314 | -0.053773269 | -0.042007187 | -0.19190020 | 0.04671794 |
| I3(t) | 0.53335779 | -0.10870162 | 0.149588405 | 0.042407001 | -0.18564054 | 0.04282109 |
| I2(t) | 0.54669516 | -0.09914391 | 0.340309073 | -0.031644459 | 0.02569882 | -0.09172520 |
| I1(t) | 0.53899832 | 0.11956319 | 0.446517814 | 0.012482056 | 0.23469771 | 0.05083778 |

**Solution for 7-stage network**

| Term-> | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Exponent = | -0.18995523 | -0.05235863 | -0.029164809 | -0.128423367 | -0.00322956 | -0.08394437 | -0.012924 |
| Frequency = | 0.16763201 | 1.29832262 | 1.593912692 | 0.55207639 | 1.95341044 | 0.94400876 | 1.8158576 |
| Coefficients of cosine terms | | | | | | | |
| I7(t) | 0.08836177 | -0.08510457 | 0.049513977 | -0.162907261 | 0.00573095 | 0.12697217 | -0.022567 |
| I6(t) | 0.06089440 | 0.07321982 | -0.082053817 | -0.075419721 | -0.01622648 | -0.01357191 | 0.0531577 |
| I5(t) | 0.00515714 | 0.09958256 | 0.00329922 | 0.0659263 | 0.02398353 | -0.14858467 | -0.0493641 |
| I4(t) | -0.08035473 | -0.06086896 | 0.082196548 | 0.204232329 | -0.02768573 | -0.13082195 | 0.0133025 |
| I3(t) | -0.19764251 | -0.11808162 | -0.057256009 | 0.27754772 | 0.02669292 | 0.03669344 | 0.0320461 |
| I2(t) | -0.34888378 | 0.04403165 | -0.048750568 | 0.240313728 | -0.02115084 | 0.19168642 | -0.0572466 |
| I1(t) | -0.53598401 | 0.13995848 | 0.09228339 | 0.082064584 | 0.01197122 | 0.16442130 | 0.045285 |
| Coefficients of sine terms | | | | | | | |
| I7(t) | 0.44237498 | -0.11107863 | 0.062564814 | -0.28577112 | 0.00707405 | 0.17848141 | -0.0280666 |
| I6(t) | 0.45153357 | 0.06428508 | -0.091728059 | -0.226484602 | -0.01984683 | 0.04080857 | 0.0634146 |
| I5(t) | 0.46817460 | 0.14141815 | -0.020687265 | -0.112597971 | 0.02875912 | -0.13509436 | -0.051698 |
| I4(t) | 0.48888126 | -0.00590248 | 0.103199858 | 0.043098464 | -0.03207143 | -0.21510853 | 0.0013296 |
| I3(t) | 0.50837302 | -0.15156538 | -0.027368426 | 0.215329738 | 0.02912692 | -0.12567774 | 0.0505977 |
| I2(t) | 0.51933597 | -0.05821287 | -0.093752148 | 0.364838351 | -0.02048050 | 0.08068096 | -0.0654593 |
| I1(t) | 0.51222581 | 0.13909249 | 0.073434672 | 0.443241858 | 0.00779474 | 0.24608923 | 0.0316274 |

**Solution for 8-stage network**

| Term-> | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Exponent = | -0.17337820 | -0.05602035 | -0.035354416 | -0.121242688 | -0.00877436 | -0.08326055 | -0.0197764 | -0.002193001 |
| Frequency = | 0.14815989 | 1.16372067 | 1.44942347 | 0.487270023 | 1.85715197 | 0.83771917 | 1.6835015 | 1.963958125 |
| Coefficients of cosine terms | | | | | | | | |
| I8(t) | 0.08244230 | -0.09099540 | 0.059942192 | -0.154747132 | 0.01564168 | 0.12621895 | -0.0345465 | -0.003956117 |
| I7(t) | 0.06201016 | 0.04722418 | -0.073545325 | -0.088261995 | -0.03892143 | 0.01399248 | 0.0661578 | 0.0113441 |
| I6(t) | 0.02059150 | 0.11782256 | -0.044853887 | 0.024928855 | 0.04224484 | -0.11789654 | -0.0256092 | -0.017228087 |
| I5(t) | -0.04287616 | 0.01067584 | 0.085645274 | 0.14985111 | -0.02379294 | -0.15658742 | -0.0437414 | 0.020825688 |
| I4(t) | -0.12986854 | -0.12112147 | 0.028271686 | 0.244719799 | -0.00712749 | -0.05952010 | 0.066301 | -0.021654888 |
| I3(t) | -0.24212693 | -0.07753382 | -0.096191594 | 0.271611642 | 0.03471826 | 0.10585293 | -0.0159286 | 0.019598076 |
| I2(t) | -0.38145102 | 0.09071176 | -0.010166406 | 0.207327682 | -0.04467891 | 0.20705074 | -0.0538752 | -0.01491867 |
| I1(t) | -0.54942444 | 0.13889111 | 0.105096303 | 0.052018069 | 0.03155979 | 0.14760946 | 0.0660217 | 0.008227976 |
| Coefficients of sine terms | | | | | | | | |
| I8(t) | 0.41071491 | -0.11715392 | 0.074489247 | -0.270980685 | 0.01889998 | 0.17580444 | -0.042155 | -0.00475666 |
| I7(t) | 0.41828076 | 0.02926927 | -0.075763473 | -0.228908817 | -0.04577507 | 0.07125557 | 0.0750028 | 0.013556304 |
| I6(t) | 0.43242409 | 0.14230382 | -0.074482279 | -0.146280237 | 0.04615672 | -0.08085268 | -0.0159761 | -0.020321396 |
| I5(t) | 0.45113170 | 0.07843279 | 0.078583362 | -0.028124834 | -0.01972630 | -0.19322748 | -0.0633874 | 0.024034702 |
| I4(t) | 0.47129459 | -0.09001741 | 0.0754346 | 0.114000663 | -0.01835000 | -0.19318352 | 0.065915 | -0.024134701 |
| I3(t) | 0.48860700 | -0.15263672 | -0.083197541 | 0.259649609 | 0.04608199 | -0.06721063 | 0.0128432 | 0.020599961 |
| I2(t) | 0.49744199 | -0.01913659 | -0.077008309 | 0.379558709 | -0.04728838 | 0.12022922 | -0.0776842 | -0.013953221 |
| I1(t) | 0.49071336 | 0.15204650 | 0.089823985 | 0.43942481 | 0.02097976 | 0.25301202 | 0.0483414 | 0.00518443 |

**Solution for 9-stage network**

| Term-> | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| Exponent = | -0.15973043 | -0.11464328 | -0.014025817 | -0.081589513 | -0.05778834 | -0.03941929 | -0.0015565 | -0.02501988 | -0.00622697 |
| Frequency = | 0.13279477 | 0.43602509 | 1.74664331 | 0.752286042 | 1.05218903 | 1.32341676 | 1.9712954 | 1.557249528 | 1.88601592 |
| Coefficients of cosine terms | | | | | | | | | |
| I9(t) | 0.07735645 | -0.14716995 | 0.024998169 | 0.123973782 | -0.09379849 | 0.06668306 | 0.0028431 | -0.043615243 | -0.01127086 |
| I8(t) | 0.06164807 | -0.09534649 | -0.052717815 | 0.033560478 | 0.02428024 | -0.05854761 | -0.0082257 | 0.066216103 | 0.02913274 |
| I7(t) | 0.02983976 | -0.00396213 | 0.033313899 | -0.086780006 | 0.11543703 | -0.07564647 | 0.0127299 | 0.009951389 | -0.03488332 |
| I6(t) | -0.01882855 | 0.10465724 | 0.016226733 | -0.154089514 | 0.06416257 | 0.05022841 | -0.0158736 | -0.072571698 | 0.02608844 |
| I5(t) | -0.08543815 | 0.20224911 | -0.051801541 | -0.116276201 | -0.07297502 | 0.08600427 | 0.0173195 | 0.027254269 | -0.00633620 |
| I4(t) | -0.17131802 | 0.26013022 | 0.041161106 | 0.011686018 | -0.12898024 | -0.04041692 | -0.0169099 | 0.060859077 | -0.01621130 |
| I3(t) | -0.27794175 | 0.25531701 | 0.007006899 | 0.152109113 | -0.02597526 | -0.09777069 | 0.014685 | -0.05959266 | 0.03216232 |
| I2(t) | -0.40679085 | 0.17626924 | -0.04995736 | 0.209535496 | 0.12194176 | 0.02768785 | -0.0108781 | -0.033028831 | -0.03477923 |
| I1(t) | -0.55917965 | 0.02717356 | 0.048519575 | 0.13147337 | 0.13355779 | 0.11035044 | 0.0058919 | 0.079404151 | 0.02280887 |
| Coefficients of sine terms | | | | | | | | | |
| I9(t) | 0.38464979 | -0.25784615 | 0.029745496 | 0.1717011 | -0.11962564 | 0.08188373 | 0.0033494 | -0.052493782 | -0.01332369 |
| I8(t) | 0.39096226 | -0.22692715 | -0.059770291 | 0.090891235 | 0.00100593 | -0.05444535 | -0.0096491 | 0.071373457 | 0.03380418 |
| I7(t) | 0.40297054 | -0.16538000 | 0.030464175 | -0.036632253 | 0.12347987 | -0.10161034 | 0.0147982 | 0.027362729 | -0.03862572 |
| I6(t) | 0.41941985 | -0.07496087 | 0.029397915 | -0.154321028 | 0.12369952 | 0.02113770 | -0.0181823 | -0.082210746 | 0.02551738 |
| I5(t) | 0.43837516 | 0.03918752 | -0.060553581 | -0.204617147 | -0.00481298 | 0.11213799 | 0.0193963 | 0.001872414 | -0.00049251 |
| I4(t) | 0.45716007 | 0.16662050 | 0.031679547 | -0.154749265 | -0.13688748 | 0.01588384 | -0.0182929 | 0.083539866 | -0.02489936 |
| I3(t) | 0.47227933 | 0.29057229 | 0.02928886 | -0.016899066 | -0.13355584 | -0.11238205 | 0.0150004 | -0.032584335 | 0.03888047 |
| I2(t) | 0.47932880 | 0.38862534 | -0.062106119 | 0.149074904 | 0.01403088 | -0.05419409 | -0.0099077 | -0.074354793 | -0.03488276 |
| I1(t) | 0.47289790 | 0.43552397 | 0.033510005 | 0.257396655 | 0.16096000 | 0.10171576 | 0.0036189 | 0.061566765 | 0.01461568 |

**Solution for 10-stage network**

| Term-> | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Exponent = | -0.14827732 | -0.10865540 | -0.018363335 | -0.010306625 | -0.00457747 | -0.04199225 | -0.0288437 | -0.079447915 | -0.05839171 | -0.00114424 |
| Frequency = | 0.12035553 | 0.39451177 | 1.636614904 | 1.792732698 | 1.90696484 | 1.21456130 | 1.4423313 | 0.682301378 | 0.95893832 | 1.97660299 |
| Coefficients of cosine terms | | | | | | | | | | |
| I10(t) | 0.07295280 | -0.14026281 | 0.032652761 | -0.018645871 | 0.00838252 | 0.07087186 | -0.0501331 | 0.12103264 | -0.09474017 | -0.00211063 |
| I9(t) | 0.06055350 | -0.09898099 | -0.057105623 | 0.042082255 | -0.02227052 | -0.04235020 | 0.0590872 | 0.047340196 | 0.00549761 | 0.00614658 |
| I8(t) | 0.03547187 | -0.02436008 | 0.009831051 | -0.034188033 | 0.02850880 | -0.08974530 | 0.0404193 | -0.059308806 | 0.10301391 | -0.00964277 |
| I7(t) | -0.00284595 | 0.06884187 | 0.050406008 | 0.000683198 | -0.02493661 | 0.00613575 | -0.0672794 | -0.138449457 | 0.09515278 | 0.01229184 |
| I6(t) | -0.05519954 | 0.16120464 | -0.047784868 | 0.033620835 | 0.01274978 | 0.09629860 | -0.0299384 | -0.141677295 | -0.01541329 | -0.01386042 |
| I5(t) | -0.12259529 | 0.23161924 | -0.015592671 | -0.04307921 | 0.00389556 | 0.03521535 | 0.0746963 | -0.059485362 | -0.11888345 | 0.01420951 |
| I4(t) | -0.20619176 | 0.26077060 | 0.060669557 | 0.020287031 | -0.01929049 | -0.08700732 | 0.0186749 | 0.070722815 | -0.10532849 | -0.01330680 |
| I3(t) | -0.30722753 | 0.23471957 | -0.029469058 | 0.018136416 | 0.02811921 | -0.07691284 | -0.0813117 | 0.180687167 | 0.02202913 | 0.01122959 |
| I2(t) | -0.42692847 | 0.14809542 | -0.040557216 | -0.043560578 | -0.02728478 | 0.05883337 | -0.0066069 | 0.204994784 | 0.14117263 | -0.00815829 |
| I1(t) | -0.56639183 | 0.00637126 | 0.060911333 | 0.036559865 | 0.01699134 | 0.11106640 | 0.0870852 | 0.116675737 | 0.12636989 | 0.00436084 |
| Coefficients of sine terms | | | | | | | | | | |
| I10(t) | 0.36272815 | -0.24620222 | 0.038409613 | -0.021765352 | 0.00973995 | 0.08626850 | -0.059722 | 0.167056903 | -0.12002413 | -0.00244637 |
| I9(t) | 0.36805270 | -0.22281505 | -0.062495231 | 0.047494755 | -0.02553298 | -0.03360991 | 0.0602978 | 0.103462232 | -0.02067342 | 0.00710194 |
| I8(t) | 0.37829918 | -0.17586540 | 0.000540377 | -0.034327926 | 0.03165581 | -0.10828753 | 0.0598454 | -0.002512316 | 0.09823297 | -0.01106893 |
| I7(t) | 0.39264923 | -0.10570885 | 0.062719682 | -0.007091444 | -0.02577386 | -0.03256917 | -0.0616917 | -0.113763105 | 0.13867926 | 0.01396242 |
| I6(t) | 0.40984286 | -0.01444588 | -0.04004528 | 0.042960656 | 0.01008760 | 0.09176234 | -0.0605396 | -0.187781213 | 0.06273009 | -0.01550123 |
| I5(t) | 0.42814041 | 0.09271522 | -0.038434354 | -0.0438111 | 0.00948815 | 0.09071448 | 0.0640129 | -0.190925754 | -0.07241555 | 0.01553507 |
| I4(t) | 0.44527365 | 0.20639787 | 0.065173114 | 0.008624618 | -0.02554689 | -0.04039960 | 0.0616664 | -0.112841855 | -0.15453099 | -0.01405923 |
| I3(t) | 0.45838736 | 0.31274989 | -0.00211748 | 0.034092362 | 0.03198240 | -0.12086420 | -0.0673608 | 0.024729044 | -0.10686792 | 0.01121510 |
| I2(t) | 0.46397375 | 0.39424087 | -0.065508409 | -0.049335178 | -0.02630121 | -0.03109344 | -0.0630776 | 0.170532964 | 0.04116953 | -0.00727669 |
| I1(t) | 0.45780233 | 0.43172327 | 0.044105603 | 0.024180163 | 0.01058303 | 0.11049152 | 0.0718249 | 0.260248668 | 0.16729904 | 0.00262435 |

## Appendix "F"

## Listing of the Visual Basic program

```
Option Strict On
Option Explicit On

' Rayleigh pulse forming network
' Jim Hawley
' August 2012

Public Class Form1
    Inherits System.Windows.Forms.Form

    Public OutputFileName As String
    Public objExcel As Microsoft.Office.Interop.Excel.Application
    Public objExcelWB As Microsoft.Office.Interop.Excel.Workbook
    Public objExcelWS As Microsoft.Office.Interop.Excel.Worksheet

    Public Sub New()
        InitializeComponent()
        With Me
            Name = ""
            Text = "Pulse Forming Network"
            FormBorderStyle = Windows.Forms.FormBorderStyle.FixedSingle
            Size = New Drawing.Size(1000, 725)
            CenterToScreen()
            .Visible = True
            Controls.Add(buttonPolynomial) : buttonPolynomial.BringToFront()
            Controls.Add(buttonExit) : buttonExit.BringToFront()
            Controls.Add(labelDisplay) : labelDisplay.BringToFront()
            PerformLayout()
            BringToFront()
        End With
    End Sub

    Private WithEvents buttonPolynomial As New Windows.Forms.Button With _
        {.Size = New Drawing.Size(200, 30), _
         .Location = New Drawing.Point(5, 5), _
         .Text = "Start", .TextAlign = ContentAlignment.MiddleCenter}

    Private WithEvents buttonExit As New Windows.Forms.Button With _
        {.Size = New Drawing.Size(200, 30), _
         .Location = New Drawing.Point(5, 40), _
         .Text = "Quit and exit", .TextAlign = ContentAlignment.MiddleCenter}

    Private Sub buttonExit_Click() Handles buttonExit.MouseClick
        Application.Exit()
    End Sub

    Public labelDisplay As New Windows.Forms.Label With _
        {.Size = New Drawing.Size(980, 600), _
         .Location = New Drawing.Point(5, 75), _
         .Text = "", .TextAlign = ContentAlignment.TopLeft}
```

```
Private Sub buttonPolynomial_Click() Handles buttonPolynomial.MouseClick

    Dim Order As Int32 = 20

    ' Open the Excel file for the results.
    OutputFileName = "C:\PFNResults.xlsx"
    Try
        objExcel = CType(CreateObject("Excel.Application"),  _
            Microsoft.Office.Interop.Excel.Application)
        objExcel.Visible = False
        objExcelWB = CType(objExcel.Workbooks.Open(OutputFileName),  _
            Microsoft.Office.Interop.Excel.Workbook)
        objExcelWS = CType(objExcelWB.Sheets("Sheet1"),  _
            Microsoft.Office.Interop.Excel.Worksheet)
    Catch ex As Exception
        Cursor.Current = Cursors.Default
        MsgBox("Could not open the Excel file to save the output.", vbOKOnly)
        Application.Exit()
    End Try
    With objExcelWS
        .Cells(1, 1) = "Rayleigh Pulse-Forming Network"
    End With
    Dim RowNum As Int32 = 3
    Dim ColNum As Int32
    ' Perform the expansion of the coefficients.
    ExpandPolynomial(Order, XArray, objExcelWS, RowNum)
    ' Add up the coefficients to get the characteristic equations.
    ExpandCharacteristic(Order, XArray, CArray, objExcelWS, RowNum)
    ' Calculate the roots and build the MMatrix for each order.
    For I As Int32 = 2 To 10 Step 1
        For J As Int32 = 0 To (2 * I) Step 1
            AVector(J) = CArray(I, J)
        Next J
        CalculateRoots(2 * I, AVector, 0.000000000001, objExcelWS, RowNum)
        BuildMMatrix(I, XArray, Roots, MMatrixRe, MMatrixIm, objExcelWS, RowNum)
        BuildSquareMatrix(I, Roots, MMatrixRe, MMatrixIm, _
            MSquareRe, MSquareIm, objExcelWS, RowNum)
        EulerReduction(I, MSquareRe, MSquareIm, _
            EulerRe, EulerIm, objExcelWS, RowNum)
        ReSubstitutionForStageN(I, EulerRe, EulerIm, _
            SolutionRe, SolutionIm, objExcelWS, RowNum)
        CheckEulerSolution(I, EulerRe, EulerIm, _
            SolutionRe, SolutionIm, objExcelWS, RowNum)
        ReSubstitutionForAllStages(I, MMatrixRe, MMatrixIm, _
            SolutionRe, SolutionIm, objExcelWS, RowNum)
        ConvertToCosSin(I, Roots, SolutionRe, SolutionIm, _
            SolutionExp, SolutionW, _
            SolutionCosRe, SolutionSinRe, SolutionCosIm, SolutionSinIm, _
            objExcelWS, RowNum)
    Next I
    ' Plot networks 2 through 10.
    ColNum = 1
    For Iplot As Int32 = 2 To Order Step 1
        AddPlotColumns(Iplot, SolutionExp, SolutionW, _
            SolutionCosRe, SolutionSinRe, objExcelWS, RowNum, ColNum)
    Next Iplot
```

```
        ' Save and close the output file.
        objExcelWB.Save()
        objExcelWB.Close()
        objExcel.Quit()
        ' All finished.
        MsgBox("All done.")
    End Sub

End Class
```

```
Option Strict Off
Option Explicit On

Public Module Solution

    '////////////////////////////////////////////////////////////////////
    '// Description of principal subroutines:

    ' ExpandPolynomial()
    ' Sub ExpandPolynomial expands the recursion to find the coefficients
    ' of the characteristic equation for all stages except the first.
    ' The coefficients are stored in XArray(,).  The results are written
    ' to the Excel file opened by Form1.  This subroutine begins its
    ' output at the row RowNum in the spreadsheet which is passed as an
    ' argument.  RowNum is returned with the new updated value.

    ' ExpandCharacteristic()
    ' Sub ExpandCharacteristic adds up the coefficients' x-powers in
    ' array XArray(,) to determine the coefficients of the charcteristic
    ' equation itself.  The coefficients are stored in CArray(,).  This
    ' subroutine can be called only after array XArray(,) has been
    ' populated to a sufficiently high power by a preceding call to the
    ' subroutine ExpandPolynomial.

    ' CalculateRoots()
    ' Sub CalculateRoots finds the roots of a polynomial of order M.
    ' Each separate root is saved in a new row of variable Roots(,).

    ' BuildMMatrix()
    ' Sub BuildMMatrix builds the MMatrix(,) for a network.  It requires
    ' that the roots for the appropriate order are known.  The roots are
    ' passed as an argument.

    ' BuildSquareMatrix()
    ' Sub BuildSquareMatrix builds the MSquare matrix.  Both the MMatrix(,)
    ' and the roots must be known, and both are passed as arguments.

    ' EulerReduction()
    ' Sub EulerReduction reduces the MSquare(,) matrix to upper triangular
    ' form.  The LHS is stored in column Euler(,2N+1).

    ' ReSubstitutionForStageN()
    ' Sub ReSubstitutionForStageN produces the coefficients for the solution
    ' for the current in the last stage in a network.  The first row of
    ' Solution(,) holds the solution for the current in stage #N.

    ' CheckEulerSolution()
    ' Sub CheckEulerSolution multiplies the upper triangular matrix by the
    ' solution for stage #N.  The errors from the expected result are stored in
    ' column Euler(,2N+2).

    ' ReSubstitutionForAllStages()
    ' Sub ReSubstitutionForAllStages produces the coefficients for the
    ' remaining stages in a network.  Each row of Solution(,) corresponds to a
    ' stage.  The first row is the solution for the current in stage #N; the
    ' last row is the solution for the current in stage #1.  Each column of
    ' Solution(,) corresponds to a single root.
```

```
' CheckFullSolution()
' Sub CheckFullSolution multiplies the MMtarix(,) by the full solution.
' This should return the initial conditions.

' ConvertToCosSin()
' Sub CovertToCosSin produces the coefficients for the solution in terms
' of cosine and sine functions.

'/////////////////////////////////////////////////////////////////
'// Description of secondary subroutines:

' FactorOneQuadratic()
' Sub FactorOneQuadratic factors one quadratic equation out of a
' polynomial of even order M.

' FactorAllQuadratics()
' Sub FactorAllQuadratics factors all quadratic equations out of a
' polynomial of even order M.  The results are stored in variable
' Quad(,).

' Sub ComplexAdd adds two complex numbers
' Sub ComplexMul multiplies two complex numbers
' Sub ComplexSub subtracts two complex number
' Sub ComplexDiv divides two complex numbers
' Sub ComplexPwr takes an integer power of a complex number

' Read MMatrixFromFile retrieves the MMatrix for a specified order
' from the Excel file.  The MSquare(,) matrix must be known and is
' passed as an argument.

'/////////////////////////////////////////////////////////////////
'// Principal variables:

Public XArray(100, 100) As Double
' This is the main repository for the coefficients.
' Row 1 contains the coefficients for the last stage
' Row 2 contains the coefficients for the second-to-last stage
' Row 3 contains the coefficients for the third-to-last stage
' ... etc
' Column 0 is the coefficient of the constant in the polynomial
' Column 1 is the coefficient of x in the polynomial
' Column 2 is the coefficient of x^2 in the polynomial
' Column 3 is the coefficient of x^3 in the polynomial
' ... etc

Public CArray(100, 100) As Double
' Row 1 contains the coefficients for a 1-stage network
' Row 2 contains the coefficients for a 2-stage network
' Row 3 contains the coefficients for a 3-stage network
' ... etc
' Column 0 is the coefficient of the constant in the polynomial
' Column 1 is the coefficient of x in the polynomial
' Column 2 is the coefficient of x^2 in the polynomial
' Column 3 is the coefficient of x^3 in the polynomial
' ... etc
```

```
Public Roots(100, 2) As Double
' Roots(,) are the roots of some polynomial.  Each row represents
' one complex conjugate root.  If the polynomial has order N, then
' N rows will be filled.

Public Quad(100, 1) As Double
' Quad() are the quadratic equations factored from some polynomial.
' The quadratic represented by row i is x^2 + Quad(i,1)x + Quad(i,0).
' If the polynomial has order N, then N/2 rows will be filled.

Public AVector(100) As Double
Public BVector(100) As Double
Public Pcoef As Double
Public Qcoef As Double
' AVector() are the coefficients for a network of a specific order
' BVector() are the coeffiencts of a polynomial reduced by two
' Pcoef is the x-coeffieicnt of a factored quadratic
' Qcoef is the constant-coefficient of a factored quadratic

Public MMatrixRe(100, 100) As Double
Public MMatrixIm(100, 100) As Double
Public MInvRe(100, 100) As Double
Public MInvIm(100, 100) As Double

Public MSquareRe(100, 100) As Double
Public MSquareIm(100, 100) As Double

Public EulerRe(100, 100) As Double
Public EulerIm(100, 100) As Double
' Euler(,) is the upper triangular verson of MSquare(,) after an
' Euler reduction by successive substitution.  Column 2N+1 is used
' to hold the cumulative factor by which each row was processed
' during the reduction.  Column 2N+2 is used to hold the errors
' which are calculated after the solution for stage N has been
' calculated.

Public SolutionRe(100, 100) As Double
Public SolutionIm(100, 100) As Double '
' Solution(,) is the solution for the coefficients of the current
' in all stages of a network.  The common factor of all terms
' is V(0)/L.  The last stage current is in column 1, the second-to-
' last stage current is in column 2, and so forth.

Public SolutionExp(100) As Double
Public SolutionW(100) As Double
Public SolutionCosRe(100, 100) As Double
Public SolutionSinRe(100, 100) As Double
Public SolutionCosIm(100, 100) As Double
Public SolutionSinIm(100, 100) As Double
' This is the final solution for the network with N stages.
' SolutionExp() holds the real exponent for each of the 2N roots.
' SolutionW() holds the angular frequency for each of the roots.
' SolutionCos() and SolutionSin() hold the coefficients of the
' cosine and sine terms for the various stages.  The first row
' is the last stage and the first column is the first root.
```

```
Public Sub ExpandPolynomial(ByVal N As Int32, _
    ByRef XArray(,) As Double, _
    ByRef objExcelWS As Microsoft.Office.Interop.Excel.Worksheet, _
    ByRef RowNum As Int32)
    '///////////////////////////////////
    '// Write the header for the output.
    With objExcelWS
        .Cells(RowNum, 1) = "Coefficients in the characteristic " & _
            "polynomials up to " & Trim(Str(N)) & " stages:"
        .Cells(RowNum + 1, 1) = "Power of x ->"
        For I As Int32 = 0 To (2 * N) Step 1
            .Cells(RowNum + 1, 2 + I) = Trim(Str(I))
        Next I
        .Cells(RowNum + 2, 1) = "Stage number"
    End With
    '////////////////
    '// Clear XArray.
    For I As Int32 = 0 To (2 * N) Step 1
        For J As Int32 = 0 To (2 * N) Step 1
            XArray(I, J) = 0
        Next J
    Next I
    '////////////////////////////////////////////
    '// Populate the last stage (1st row) in XArray.
    XArray(1, 0) = 1
    '//////////////////////////////////////////////////////////
    '// Populate the second-to-last stage (2nd row) in XArray.
    XArray(2, 0) = 1
    XArray(2, 2) = 1
    '//////////////////////////////////////////////////////////
    '// Main loop to step through the remaining rows in XArray.
    '// Ai = (1+x^2)Ai-1 + x^2(Sum of Aj up to i-2)
    For I As Int32 = 3 To N Step 1
        ' First pass - Add the following stage times 1.
        For J As Int32 = 0 To (2 * N) Step 1
            XArray(I, J) = XArray(I - 1, J)
        Next J
        ' Second pass - Add the following stage times x^2.
        For J As Int32 = 0 To (2 * N) Step 1
            XArray(I, J + 2) = XArray(I, J + 2) + XArray(I - 1, J)
        Next J
        ' Sub-loop to step through the second-last stages times x^2.
        For J As Int32 = 1 To (I - 2) Step 1
            ' Third pass - Add the following stage times x^2.
            For K As Int32 = 0 To (2 * N) Step 1
                XArray(I, K + 2) = XArray(I, K + 2) + XArray(J, K)
            Next K
        Next J
    Next I
    '////////////////////////////////////
    '// Write the coefficients to the file.
    For I As Int32 = 1 To N Step 1
        With objExcelWS
            .Cells(RowNum + 2 + I, 1) = Trim(Str(N + 1 - I))
            For J As Int32 = 0 To (2 * N) Step 1
                .Cells(RowNum + 2 + I, 2 + J) = Trim(Str(XArray(I, J)))
            Next J
        End With
```

```
    Next I
    '////////////////////////////////////////////
    '// Update RowNum to the end of the spreadsheet.
    RowNum = RowNum + 4 + N
End Sub

Public Sub ExpandCharacteristic(ByVal N As Int32, _
    ByVal XArray(,) As Double, _
    ByRef CArray(,) As Double, _
    ByRef objExcelWS As Microsoft.Office.Interop.Excel.Worksheet, _
    ByRef RowNum As Int32)
    '////////////////////////////////////
    '// Write the header for the output.
    With objExcelWS
        .Cells(RowNum, 1) = "Coefficients in the characteristic " & _
            "equations up to " & Trim(Str(N)) & " stages:"
        .Cells(RowNum + 1, 1) = "Power of x ->"
        For I As Int32 = 0 To (2 * N) Step 1
            .Cells(RowNum + 1, 2 + I) = Trim(Str(I))
        Next I
        .Cells(RowNum + 2, 1) = "Number of stages"
    End With
    '////////////////
    '// Clear CArray.
    For I As Int32 = 0 To (2 * N) Step 1
        For J As Int32 = 0 To (2 * N) Step 1
            CArray(I, J) = 0
        Next J
    Next I
    '////////////////////////////////////////////////
    '// Populate the 1-stage network (1st row) in CArray.
    CArray(1, 0) = 1
    CArray(1, 1) = 1
    CArray(1, 2) = 1
    '//////////////////////////////////////////////////////////
    '// Main loop to step through the remaining rows in CArray.
    '// x^2(Sum of Ai from 1 to N) + x(Sum of Ai from 1 to N) + A1
    For I As Int32 = 2 To N Step 1
        ' Sub-loop to step through the x^2 terms.
        For J As Int32 = 1 To I Step 1
            ' First pass - Add stage #J times x^2.
            For K As Int32 = 0 To (2 * N) Step 1
                CArray(I, K + 2) = CArray(I, K + 2) + XArray(J, K)
            Next K
        Next J
        ' Sub-loop to step through the x terms.
        For J As Int32 = 1 To I Step 1
            ' Second pass - Add stage #J times x.
            For K As Int32 = 0 To (2 * N) Step 1
                CArray(I, K + 1) = CArray(I, K + 1) + XArray(J, K)
            Next K
        Next J
        ' Third pass - Add A1.
        For J As Int32 = 0 To (2 * N) Step 1
            CArray(I, J) = CArray(I, J) + XArray(I, J)
        Next J
    Next I
```

```
    '////////////////////////////////////
    '// Write the coefficients to the file.
    For I As Int32 = 1 To N Step 1
        With objExcelWS
            .Cells(RowNum + 2 + I, 1) = Trim(Str(I))
            For J As Int32 = 0 To (2 * N) Step 1
                .Cells(RowNum + 2 + I, 2 + J) = Trim(Str(CArray(I, J)))
            Next J
        End With
    Next I
    '////////////////////////////////////////////
    '// Update RowNum to the end of the spreadsheet.
    RowNum = RowNum + 4 + N
End Sub

Public Sub CalculateRoots( _
    ByVal M As Int32, _
    ByVal A() As Double, _
    ByVal MaxErr As Double, _
    ByRef objExcelWS As Microsoft.Office.Interop.Excel.Worksheet, _
    ByRef RowNum As Int32)
    ' This subroutine factors a polynomial of even order M, whose
    ' coefficients are passed in as vector A(), into its roots.  There
    ' will be M/2 complex conjugate pairs of roots.  They are stored
    ' individually in the first M rows of Roots(,2).  Roots(,1) is the
    ' real part of the root and Roots(,2) is the imaginary part.
    '////////////////////////////////
    '// Check for an odd order of M.
    If (Math.Abs(2 * Math.Round((M + 0.1) / 2) - M) > 0.1) Then
        MsgBox("CalculateRoots received an odd-order polynomial")
        Exit Sub
    End If
    '////////////////////////////////////
    '// Write the header for the output.
    With objExcelWS
        .Cells(RowNum, 1) = "Roots of the characteristic equation " & _
            "for " & Trim(Str(CInt(M / 2))) & " stages:"
        .Cells(RowNum + 1, 2) = "Re part"
        .Cells(RowNum + 1, 3) = "Im part"
        For I As Int32 = 1 To M Step 1
            .Cells(RowNum + 1 + I, 1) = "Root " & Trim(Str(I))
        Next
    End With
    '////////////////////////////////////////////////
    '// Factor the poloynomial into quadratic equations.
    FactorAllQuadratics(M, A, 0.000000000001)
    ' Display the interim results on the screen.
    Form1.labelDisplay.Text = "Quadratic equations are:" & vbCrLf
    For I As Int32 = 1 To CInt(M / 2) Step 1
        Form1.labelDisplay.Text = Form1.labelDisplay.Text & _
             "x^2 + " & Trim(Str(Quad(I, 1))) & _
             "x + " & Trim(Str(Quad(I, 0))) & vbCrLf
    Next I
    Form1.labelDisplay.Refresh()
    ' MsgBox("OK to proceed?")
```

```
'////////////////////////////////////////////////
'// Main loop to step through the quadratic equations.
Dim AA As Double
Dim BB As Double
Dim CC As Double
Dim Radical As Double
For Ieqn As Int32 = 1 To CInt(M / 2) Step 1
    Dim Iroot As Int32
    Iroot = (2 * Ieqn) - 1
    AA = 1
    BB = Quad(Ieqn, 1)
    CC = Quad(Ieqn, 0)
    Roots(Iroot, 1) = -BB / (2 * AA)
    Roots(Iroot + 1, 1) = Roots(Iroot, 1)
    Radical = ((4 * AA * CC) - (BB * BB)) / (2 * AA)
    If (Radical < 0) Then
        MsgBox("CalculateRoots encountered a positive radical.")
        Exit Sub
    End If
    Roots(Iroot, 2) = Math.Sqrt(Radical / (2 * AA))
    Roots(Iroot + 1, 2) = -Roots(Iroot, 2)
Next Ieqn
'////////////////////////////////////////////////
'// Sort the roots in order of decreasing magnitude.
Dim TempSq(M) As Double
Dim TempRe As Double
Dim TempIm As Double
Dim MaxVal As Double
Dim MaxIndex As Int32
For I As Int32 = 1 To M Step 1
    ComplexMul(Roots(I, 1), Roots(I, 2), _
        Roots(I, 1), -Roots(I, 2), TempRe, TempIm)
    TempSq(I) = TempRe
Next I
For I As Int32 = 1 To M Step 1
    MaxVal = -1
    For J As Int32 = 1 To M Step 1
        ' Find the root with the greatest magnitude.
        If (TempSq(J) > MaxVal) Then
            MaxVal = TempSq(I)
            MaxIndex = I
        End If
        ' Swap the roots.
        TempRe = Roots(MaxIndex, 1)
        TempIm = Roots(MaxIndex, 2)
        Roots(MaxIndex, 1) = Roots(I, 1)
        Roots(MaxIndex, 2) = Roots(I, 2)
        Roots(I, 1) = TempRe
        Roots(I, 2) = TempIm
    Next J
Next I
'////////////////////////////////
'// Write the roots to the file.
For I As Int32 = 1 To M Step 1
    Dim Re As Double
    Dim Im As Double
    Re = Roots(I, 1)
    Im = Roots(I, 2)
```

```
        With objExcelWS
            .Cells(RowNum + 1 + I, 2) = Trim(Str(Roots(I, 1)))
            .Cells(RowNum + 1 + I, 3) = Trim(Str(Roots(I, 2)))
        End With
    Next I
    '////////////////////////////////////////////
    '// Update RowNum to the end of the spreadsheet.
    RowNum = RowNum + 3 + M
End Sub

Public Sub BuildMMatrix( _
    ByVal N As Int32, _
    ByVal XArray(,) As Double, _
    ByVal Roots(,) As Double, _
    ByRef MMatrixRe(,) As Double, _
    ByRef MMatrixIm(,) As Double, _
    ByRef objExcelWS As Microsoft.Office.Interop.Excel.Worksheet, _
    ByRef RowNum As Int32)
    Dim TempRe As Double
    Dim TempIm As Double
    '////////////////////////////////////
    '// Write the header for the output.
    With objExcelWS
        .Cells(RowNum, 1) = "MMatrix for " & Trim(Str(N)) & " stages:"
        .Cells(RowNum + 1, 1) = "Roots->"
        For I As Int32 = 1 To (2 * N) Step 1
            .Cells(RowNum + 1, 1 + I) = Trim(Str(I))
        Next I
        .Cells(RowNum + 2, 1) = "Re part"
        .Cells(RowNum + 3 + N, 1) = "Im part"
    End With
    '////////////////////////////////////////////
    '// Main loop to step through the rows of XArray.
    For Irow As Int32 = 1 To N Step 1
        ' Sub-loop to step through all of the roots.
        For Iroot As Int32 = 1 To (2 * N) Step 1
            MMatrixRe(Irow, Iroot) = 0
            MMatrixIm(Irow, Iroot) = 0
            ' Sub-loop to step through all of the powers of x.
            For Ipower As Int32 = 0 To ((2 * N) - 2) Step 1
                If (XArray(Irow, Ipower) <> 0) Then
                    ComplexPwr( _
                        Roots(Iroot, 1), Roots(Iroot, 2), _
                        Ipower, TempRe, TempIm)
                    MMatrixRe(Irow, Iroot) = MMatrixRe(Irow, Iroot) + _
                        XArray(Irow, Ipower) * TempRe
                    MMatrixIm(Irow, Iroot) = MMatrixIm(Irow, Iroot) + _
                        XArray(Irow, Ipower) * TempIm
                End If
            Next Ipower
        Next Iroot
    Next Irow
    '////////////////////
    '// Write the matrix.
    With objExcelWS
        For Irow As Int32 = 1 To N Step 1
            For Iroot As Int32 = 1 To (2 * N) Step 1
                .Cells(RowNum + 2 + Irow, 1) = _
```

```
                    "   Stage " & Trim(Str(N + 1 - Irow))
                .Cells(RowNum + 2 + Irow, 1 + Iroot) = MMatrixRe(Irow, Iroot)
                .Cells(RowNum + 3 + N + Irow, 1) = _
                    "   Stage " & Trim(Str(N + 1 - Irow))
                .Cells(RowNum + 3 + N + Irow, 1 + Iroot) = MMatrixIm(Irow, Iroot)
            Next Iroot
        Next Irow
    End With
    '////////////////////////////////////////////
    '// Update RowNum to the end of the spreadsheet.
    RowNum = RowNum + 5 + (2 * N)
    '////////////////////////////////////////////
    ' Display the interim results on the screen.
    Form1.labelDisplay.Text = "MMatrix for order = " & _
        Trim(Str(N)) & vbCrLf & "Real parts:" & vbCrLf
    For Istage As Int32 = 1 To N Step 1
        For Iroot As Int32 = 1 To (2 * N) Step 1
            Form1.labelDisplay.Text = Form1.labelDisplay.Text & _
                Trim(Str(MMatrixRe(Istage, Iroot))) & " "
        Next Iroot
        Form1.labelDisplay.Text = Form1.labelDisplay.Text & vbCrLf
    Next Istage
    Form1.labelDisplay.Text = Form1.labelDisplay.Text & _
        "Imaginary parts:" & vbCrLf
    For Istage As Int32 = 1 To N Step 1
        For Iroot As Int32 = 1 To (2 * N) Step 1
            Form1.labelDisplay.Text = Form1.labelDisplay.Text & _
                Trim(Str(MMatrixIm(Istage, Iroot))) & " "
        Next Iroot
        Form1.labelDisplay.Text = Form1.labelDisplay.Text & vbCrLf
    Next Istage
    Form1.labelDisplay.Refresh()
End Sub

Public Sub BuildSquareMatrix( _
    ByVal N As Int32, _
    ByVal Roots(,) As Double, _
    ByVal MMatrixRe(,) As Double, _
    ByVal MMatrixIm(,) As Double, _
    ByRef MSquareRe(,) As Double, _
    ByRef MSquareIm(,) As Double, _
    ByRef objExcelWS As Microsoft.Office.Interop.Excel.Worksheet, _
    ByRef RowNum As Int32)
    Dim TempRe As Double
    Dim TempIm As Double
    '////////////////////////////////
    '// Display message on the screen.
    Form1.labelDisplay.Text = "Building the square matrix for order " & Trim(Str(N))
    Form1.labelDisplay.Refresh()
    '////////////////////////////////
    '// Write the header for the output.
    With objExcelWS
        .Cells(RowNum, 1) = "SquareMatrix for " & Trim(Str(N)) & " stages:"
        .Cells(RowNum + 1, 1) = "Roots->"
        For I As Int32 = 1 To (2 * N) Step 1
            .Cells(RowNum + 1, 1 + I) = Trim(Str(I))
        Next I
        .Cells(RowNum + 1, 2 + (2 * N)) = "LHS"
```

```
        .Cells(RowNum + 2, 1) = "Re part"
        .Cells(RowNum + 3 + (2 * N), 1) = "Im part"
    End With
    '//////////////////////////////////////
    '// Main loop to step through the 2N roots.
    For Iroot As Int32 = 1 To (2 * N) Step 1
        ' Sub-loop to step through the first N equations.
        For Ieqn As Int32 = 1 To N Step 1
            MSquareRe(Ieqn, Iroot) = MMatrixRe(Ieqn, Iroot)
            MSquareIm(Ieqn, Iroot) = MMatrixIm(Ieqn, Iroot)
        Next Ieqn
        ' Sub-loop to step through the second N equations.
        For Ieqn As Int32 = 1 To N Step 1
            ComplexMul(MSquareRe(Ieqn, Iroot), MSquareIm(Ieqn, Iroot), _
                Roots(Iroot, 1), Roots(Iroot, 2), TempRe, TempIm)
            MSquareRe(N + Ieqn, Iroot) = TempRe
            MSquareIm(N + Ieqn, Iroot) = TempIm
        Next Ieqn
    Next Iroot
    '////////////////////
    '// Write the matrix.
    With objExcelWS
        For Ieqn As Int32 = 1 To (2 * N) Step 1
            For Iroot As Int32 = 1 To (2 * N) Step 1
                .Cells(RowNum + 2 + Ieqn, 1) = "   Eqn " & Trim(Str(Ieqn))
                .Cells(RowNum + 2 + Ieqn, 1 + Iroot) = _
                    MSquareRe(Ieqn, Iroot)
                .Cells(RowNum + 3 + (2 * N) + Ieqn, 1) = "   Eqn " & Trim(Str(Ieqn))
                .Cells(RowNum + 3 + (2 * N) + Ieqn, 1 + Iroot) = _
                    MSquareIm(Ieqn, Iroot)
            Next Iroot
            .Cells(RowNum + 2 + Ieqn, 2 + (2 * N)) = 0
            .Cells(RowNum + 3 + (2 * N) + Ieqn, 2 + (2 * N)) = 0
        Next Ieqn
        .Cells(RowNum + 2 + (2 * N), 2 + (2 * N)) = 1
    End With
    '//////////////////////////////////////////////
    '// Update RowNum to the end of the spreadsheet.
    RowNum = RowNum + 5 + (4 * N)
End Sub

Public Sub EulerReduction( _
    ByVal N As Int32, _
    ByVal MSquareRe(,) As Double, _
    ByVal MSquareIm(,) As Double, _
    ByRef EulerRe(,) As Double, _
    ByRef EulerIm(,) As Double, _
    ByRef objExcelWS As Microsoft.Office.Interop.Excel.Worksheet, _
    ByRef RowNum As Int32)
    Dim LeadCoefRe As Double
    Dim LeadCoefIm As Double
    Dim TempRe As Double
    Dim TempIm As Double
    '//////////////////////////////////
    '// Display message on the screen.
    Form1.labelDisplay.Text = "Reducing the square matrix for order " & Trim(Str(N))
    Form1.labelDisplay.Refresh()
```

```
'////////////////////////////////
'// Write the header for the output.
With objExcelWS
    .Cells(RowNum, 1) = "Euler matrix for " & Trim(Str(N)) & " stages:"
    .Cells(RowNum + 1, 1) = "Roots->"
    For I As Int32 = 1 To (2 * N) Step 1
        .Cells(RowNum + 1, 1 + I) = Trim(Str(I))
    Next I
    .Cells(RowNum + 1, 2 + (2 * N)) = "LHS"
    .Cells(RowNum + 2, 1) = "Re part"
    .Cells(RowNum + 3 + (2 * N), 1) = "Im part"
End With
'//////////////////////////////////////
'// Copy matrix MSquare(,) into Euler(,).
For Irow As Int32 = 1 To (2 * N) Step 1
    For Icol As Int32 = 1 To (2 * N) Step 1
        EulerRe(Irow, Icol) = MSquareRe(Irow, Icol)
        EulerIm(Irow, Icol) = MSquareIm(Irow, Icol)
    Next Icol
Next Irow
'//////////////////////////////////////////////////////////////////////
'// Initialize the LHS column to unity and the error column to zero.
For Irow As Int32 = 1 To (2 * N) Step 1
    EulerRe(Irow, (2 * N) + 1) = 0
    EulerIm(Irow, (2 * N) + 1) = 0
    EulerRe(Irow, (2 * N) + 2) = 0
    EulerIm(Irow, (2 * N) + 2) = 0
Next Irow
EulerRe(2 * N, (2 * N) + 1) = 1
'//////////////////////////////////////////
'// Main loop to step from row #2 to row #2N.
For Imain As Int32 = 2 To (2 * N) Step 1
    ' Step #1 - Divide rows #Imain to row #2N by the leading
    ' coefficient of the row.
    For Irow As Int32 = (Imain - 1) To (2 * N) Step 1
        LeadCoefRe = EulerRe(Irow, Imain - 1)
        LeadCoefIm = EulerIm(Irow, Imain - 1)
        For Icol As Int32 = (Imain - 1) To ((2 * N) + 1) Step 1
            ComplexDiv(EulerRe(Irow, Icol), EulerIm(Irow, Icol), _
                LeadCoefRe, LeadCoefIm, TempRe, TempIm)
            EulerRe(Irow, Icol) = TempRe
            EulerIm(Irow, Icol) = TempIm
        Next Icol
    Next Irow
    ' Step #2 - Subtract row #Imain-1 from rows #Imain to row #2N.
    For Irow As Int32 = Imain To (2 * N) Step 1
        For Icol As Int32 = (Imain - 1) To (2 * N) Step 1
            ComplexSub(EulerRe(Irow, Icol), EulerIm(Irow, Icol), _
                EulerRe(Imain - 1, Icol), EulerIm(Imain - 1, Icol), _
                TempRe, TempIm)
            EulerRe(Irow, Icol) = TempRe
            EulerIm(Irow, Icol) = TempIm
        Next Icol
    Next Irow
Next Imain
```

```
    '////////////////////
    '// Write the matrix.
    With objExcelWS
        For Irow As Int32 = 1 To (2 * N) Step 1
            For Icol As Int32 = 1 To ((2 * N) + 1) Step 1
                .Cells(RowNum + 2 + Irow, 1) = "   Eqn " & Trim(Str(Irow))
                .Cells(RowNum + 2 + Irow, 1 + Icol) = EulerRe(Irow, Icol)
                .Cells(RowNum + 3 + (2 * N) + Irow, 1) = "   Eqn " & Trim(Str(Irow))
                .Cells(RowNum + 3 + (2 * N) + Irow, 1 + Icol) = EulerIm(Irow, Icol)
            Next Icol
        Next Irow
    End With
    '////////////////////////////////////////////
    '// Update RowNum to the end of the spreadsheet.
    RowNum = RowNum + 5 + (4 * N)
End Sub

Public Sub ReSubstitutionForStageN( _
    ByVal N As Int32, _
    ByVal EulerRe(,) As Double, _
    ByVal EulerIm(,) As Double, _
    ByRef SolutionRe(,) As Double, _
    ByRef SolutionIm(,) As Double, _
    ByRef objExcelWS As Microsoft.Office.Interop.Excel.Worksheet, _
    ByRef RowNum As Int32)
    ' The solution for the current in stage #N is stored in the
    ' first row of Solution(,).
    Dim TempRe As Double
    Dim TempIm As Double
    Dim RowSumRe As Double
    Dim RowSumIm As Double
    '////////////////////////////////
    '// Display message on the screen.
    Form1.labelDisplay.Text = "ReSubstituting for order " & Trim(Str(N))
    Form1.labelDisplay.Refresh()
    '////////////////////////////////
    '// Write the header for the output.
    With objExcelWS
        .Cells(RowNum, 1) = "Solution vector for last stage N = " & _
            Trim(Str(N)) & ":"
        .Cells(RowNum + 1, 1) = "Root->"
        For Iroot As Int32 = 1 To (2 * N) Step 1
            .Cells(RowNum + 1, Iroot + 1) = Trim(Str(Iroot))
        Next Iroot
        .Cells(RowNum + 2, 1) = "Re part"
        .Cells(RowNum + 3, 1) = "Im part"
    End With
    '////////////////////////////////////////
    '// Clear the solution matrix.
    For Irow As Int32 = 1 To N Step 1
        For Icol As Int32 = 1 To (2 * N) Step 1
            SolutionRe(Irow, Icol) = 0
            SolutionIm(Irow, Icol) = 0
        Next Icol
    Next Irow
```

```
    '////////////////////////////////////////////////
    '// Calculate the solution for the last coefficient.
    ComplexDiv(EulerRe(2 * N, (2 * N) + 1), EulerIm(2 * N, (2 * N) + 1), _
        EulerRe(2 * N, 2 * N), EulerIm(2 * N, 2 * N), TempRe, TempIm)
    SolutionRe(1, 2 * N) = TempRe
    SolutionIm(1, 2 * N) = TempIm
    '////////////////////////////////////////////////
    '// Main loop to step through the other coefficients.
    For Imain As Int32 = (2 * N) - 1 To 1 Step -1
        RowSumRe = 0
        RowSumIm = 0
        ' Step #1 - Add up the balance of the row.
        For K As Int32 = (Imain + 1) To (2 * N) Step 1
            ComplexMul(EulerRe(Imain, K), EulerIm(Imain, K), _
                SolutionRe(1, K), SolutionIm(1, K), TempRe, TempIm)
            ComplexAdd(RowSumRe, RowSumIm, TempRe, TempIm, TempRe, TempIm)
            RowSumRe = TempRe
            RowSumIm = TempIm
        Next K
        ' Step #2 - Divide by the leading coefficient.
        ComplexDiv(-RowSumRe, -RowSumIm, _
            EulerRe(Imain, Imain), EulerIm(Imain, Imain), TempRe, TempIm)
        SolutionRe(1, Imain) = TempRe
        SolutionIm(1, Imain) = TempIm
    Next Imain
    '////////////////////////////////////////
    '// Write the solution for the last stage.
    With objExcelWS
        For Iroot As Int32 = 1 To (2 * N) Step 1
            .Cells(RowNum + 2, Iroot + 1) = SolutionRe(1, Iroot)
            .Cells(RowNum + 3, Iroot + 1) = SolutionIm(1, Iroot)
        Next Iroot
    End With
    '////////////////////////////////////////////
    '// Update RowNum to the end of the spreadsheet.
    RowNum = RowNum + 5
End Sub

Public Sub CheckEulerSolution( _
    ByVal N As Int32, _
    ByRef EulerRe(,) As Double, _
    ByRef EulerIm(,) As Double, _
    ByVal SolutionRe(,) As Double, _
    ByVal SolutionIm(,) As Double, _
    ByRef objExcelWS As Microsoft.Office.Interop.Excel.Worksheet, _
    ByRef RowNum As Int32)
    Dim TempRe As Double
    Dim TempIm As Double
    Dim RowSumRe As Double
    Dim RowSumIm As Double
    '////////////////////////////////
    '// Display message on the screen.
    Form1.labelDisplay.Text = "Checking solution for order " & Trim(Str(N))
    Form1.labelDisplay.Refresh()
```

```
    '//////////////////////////////////
    '// Write the header for the output.
    With objExcelWS
        .Cells(RowNum, 1) = "Errors in solution for last stage N = " _
            & Trim(Str(N)) & ":"
        .Cells(RowNum + 1, 1) = "Root->"
        For Iroot As Int32 = 1 To (2 * N) Step 1
            .Cells(RowNum + 1, Iroot + 1) = Trim(Str(Iroot))
        Next Iroot
        .Cells(RowNum + 2, 1) = "Re part"
        .Cells(RowNum + 3, 1) = "Im part"
    End With
    '//////////////////////////////////////////
    '// Main loop to step through the equations.
    For Ieqn As Int32 = 1 To (2 * N) Step 1
        RowSumRe = 0
        RowSumIm = 0
        ' Step #1 - Multiply the row by the solution for stage N.
        For K As Int32 = 1 To (2 * N) Step 1
            ComplexMul(EulerRe(Ieqn, K), EulerIm(Ieqn, K), _
                SolutionRe(1, K), SolutionIm(1, K), TempRe, TempIm)
            RowSumRe = RowSumRe + TempRe
            RowSumIm = RowSumIm + TempIm
        Next K
        ' Step #2 - For all equations but the last, compare the sum to zero.
        If (Ieqn <> (2 * N)) Then
            EulerRe(Ieqn, (2 * N) + 2) = RowSumRe
            EulerIm(Ieqn, (2 * N) + 2) = RowSumIm
            ' Step #3 - For the last row, compare the sum to the LHS.
        Else
            ComplexDiv(RowSumRe, RowSumIm, _
                EulerRe(Ieqn, (2 * N) + 1), EulerIm(Ieqn, (2 * N) + 1), _
                TempRe, TempIm)
            EulerRe(Ieqn, (2 * N) + 2) = TempRe - 1
            EulerIm(Ieqn, (2 * N) + 2) = TempIm
        End If
    Next Ieqn
    '////////////////////
    '// Write the errors.
    With objExcelWS
        For Iroot As Int32 = 1 To (2 * N) Step 1
            .Cells(RowNum + 2, Iroot + 1) = EulerRe(Iroot, (2 * N) + 2)
            .Cells(RowNum + 3, Iroot + 1) = EulerIm(Iroot, (2 * N) + 2)
        Next Iroot
    End With
    '////////////////////////////////////////////
    '// Update RowNum to the end of the spreadsheet.
    RowNum = RowNum + 5
End Sub

Public Sub ReSubstitutionForAllStages( _
    ByVal N As Int32, _
    ByVal MMatrixRe(,) As Double, _
    ByVal MMatrixIm(,) As Double, _
    ByRef SolutionRe(,) As Double, _
    ByRef SolutionIm(,) As Double, _
    ByRef objExcelWS As Microsoft.Office.Interop.Excel.Worksheet, _
    ByRef RowNum As Int32)
```

```
    Dim TempRe As Double
    Dim TempIm As Double
    '//////////////////////////////
    '// Display message on the screen.
    Form1.labelDisplay.Text = "ReSubstituting all stages of order " & Trim(Str(N))
    Form1.labelDisplay.Refresh()
    '//////////////////////////////////
    '// Write the header for the output.
    With objExcelWS
        .Cells(RowNum, 1) = "Complete solution matrix for " & _
            Trim(Str(N)) & " stages:"
        .Cells(RowNum + 1, 1) = "Root->"
        For Iroot As Int32 = 1 To (2 * N) Step 1
            .Cells(RowNum + 1, Iroot + 1) = Trim(Str(Iroot))
        Next Iroot
        .Cells(RowNum + 2, 1) = "Re part"
        .Cells(RowNum + 3 + N, 1) = "Im part"
    End With
    '////////////////////////////////////////////
    '// Main loop to step from stage #N-1 to stage #1.
    For Istage As Int32 = 2 To N Step 1
        ' Mulitply MMAtrix(Imain,Icol) x SolutionForN(Icol).
        For K As Int32 = 1 To (2 * N) Step 1
            ComplexMul(MMatrixRe(Istage, K), MMatrixIm(Istage, K), _
                SolutionRe(1, K), SolutionIm(1, K), TempRe, TempIm)
            SolutionRe(Istage, K) = TempRe
            SolutionIm(Istage, K) = TempIm
        Next K
    Next Istage
    '////////////////////
    '// Write the matrix.
    With objExcelWS
        For Irow As Int32 = 1 To N Step 1
            .Cells(RowNum + 2 + Irow, 1) = _
                "  I" & Trim(Str(N + 1 - Irow)) & "(t)"
            .Cells(RowNum + 3 + N + Irow, 1) = _
                "  I" & Trim(Str(N + 1 - Irow)) & "(t)"
        Next Irow
        For Irow As Int32 = 1 To N Step 1
            For Iroot As Int32 = 1 To (2 * N) Step 1
                .Cells(RowNum + 2 + Irow, Iroot + 1) = SolutionRe(Irow, Iroot)
                .Cells(RowNum + 3 + N + Irow, Iroot + 1) = SolutionIm(Irow, Iroot)
            Next Iroot
        Next Irow
    End With
    '////////////////////////////////////////////
    '// Update RowNum to the end of the spreadsheet.
    RowNum = RowNum + 5 + (2 * N)
End Sub

Public Sub ConvertToCosSin( _
    ByVal N As Int32, _
    ByVal Roots(,) As Double, _
    ByVal SolutionRe(,) As Double, _
    ByVal SolutionIm(,) As Double, _
    ByRef SolutionExp() As Double, _
    ByRef SolutionW() As Double, _
    ByRef SolutionCosRe(,) As Double, _
```

```
ByRef SolutionSinRe(,) As Double, _
ByRef SolutionCosIm(,) As Double, _
ByRef SolutionSinIm(,) As Double, _
ByRef objExcelWS As Microsoft.Office.Interop.Excel.Worksheet, _
ByRef RowNum As Int32)
'////////////////////////////////
'// Display message on the screen.
Form1.labelDisplay.Text = "Converting to cos and sin for order " & Trim(Str(N))
Form1.labelDisplay.Refresh()
'////////////////////////////////
'// Write the header for the output.
With objExcelWS
    .Cells(RowNum, 1) = "Complete real matrix for " & Trim(Str(N)) & " stages:"
    .Cells(RowNum + 1, 1) = "Term->"
    .Cells(RowNum + 2, 1) = "Exponent"
    .Cells(RowNum + 3, 1) = "Frequency"
    For Icol As Int32 = 1 To N Step 1
        Dim Iroot As Int32 = (2 * Icol) - 1
        .Cells(RowNum + 1, Icol + 1) = Trim(Str(Icol))
        .Cells(RowNum + 2, Icol + 1) = Trim(Str(Roots(Iroot, 1)))
        .Cells(RowNum + 3, Icol + 1) = Trim(Str(Roots(Iroot, 2)))
    Next Icol
    .Cells(RowNum + 4, 1) = "Cosine coefficient (Real)"
    .Cells(RowNum + 5 + N, 1) = "Sine coefficient (Real)"
    .Cells(RowNum + 6 + (2 * N), 1) = "Cosine coefficient (Imaginary)"
    .Cells(RowNum + 7 + (3 * N), 1) = "Sine coefficient (Imaginary)"
End With
'////////////////////////////
'// Main loop for all stages.
For Istage As Int32 = 1 To N Step 1
    For Iroot As Int32 = 1 To N Step 1
        Dim Icol As Int32 = (2 * Iroot) - 1
        ' (A+jB)e^jx + (C+jD)e^-x = ...
        ' ... (A + C + jB + jD)cosx + (-B + D + jA - jC)sinx
        SolutionCosRe(Istage, Iroot) = _
            SolutionRe(Istage, Icol) + SolutionRe(Istage, Icol + 1)
        SolutionCosIm(Istage, Iroot) = _
            SolutionIm(Istage, Icol) + SolutionIm(Istage, Icol + 1)
        SolutionSinRe(Istage, Iroot) = _
            -SolutionIm(Istage, Icol) + SolutionIm(Istage, Icol + 1)
        SolutionSinIm(Istage, Iroot) = _
            SolutionRe(Istage, Icol) - SolutionRe(Istage, Icol + 1)
    Next Iroot
Next Istage
'////////////////////
'// Write the matrix.
With objExcelWS
    For Istage As Int32 = N To 1 Step -1
        Dim Irow As Int32 = N + 1 - Istage
        .Cells(RowNum + 4 + Irow, 1) = _
            "  I" & Trim(Str(Istage)) & "(t)"
        .Cells(RowNum + 5 + N + Irow, 1) = _
            "  I" & Trim(Str(Istage)) & "(t)"
        .Cells(RowNum + 6 + (2 * N) + Irow, 1) = _
            "  I" & Trim(Str(Istage)) & "(t)"
        .Cells(RowNum + 7 + (3 * N) + Irow, 1) = _
            "  I" & Trim(Str(Istage)) & "(t)"
    Next Istage
```

```
        For Istage As Int32 = N To 1 Step -1
            Dim Irow As Int32 = N + 1 - Istage
            For Iroot As Int32 = 1 To N Step 1
                .Cells(RowNum + 4 + Irow, Iroot + 1) = SolutionCosRe(Irow, Iroot)
                .Cells(RowNum + 5 + N + Irow, Iroot + 1) = SolutionSinRe(Irow, Iroot)
                .Cells(RowNum + 6 + (2 * N) + Irow, Iroot + 1) = _
                    SolutionCosIm(Irow, Iroot)
                .Cells(RowNum + 7 + (3 * N) + Irow, Iroot + 1) = _
                    SolutionSinIm(Irow, Iroot)
            Next Iroot
        Next Istage
    End With
    '////////////////////////////////////////////
    '// Update RowNum to the end of the spreadsheet.
    RowNum = RowNum + 9 + (4 * N)
End Sub

'////////////////////////////////////////////////////////////////////////////////
'////////////////////////////////////////////////////////////////////////////////
'
' Secondary subroutines
'
'////////////////////////////////////////////////////////////////////////////////
'////////////////////////////////////////////////////////////////////////////////

Public Sub FactorOneQuadratic(ByVal M As Int32, _
    ByVal A() As Double, _
    ByRef P As Double, ByRef Q As Double, _
    ByRef B() As Double, _
    ByVal MaxErr As Double)
    ' This subroutine factors one quadratic equation out of the polynomial A()
    ' passed as an argument.  The values in A() are the coefficients Ai of the
    ' powers from i=0 to i=M.  The returned vector B() contains the coefficients
    ' Bi of the powers of the reduced polynomial, from i=0 to i=M-2.  The
    ' coefficients of the extracted quadratic are P and Q, and are returned
    ' individually.  The expansion takes the following form:
    ' Sum(Ai x^i) = (x^2 + Px + Q)Sum(Bi x^i) + Rx + S = 0.
    ' The argument MaxErr is the maximum fractional error acceptable in R and S,
    ' and in delta(P) and delata(Q), in the final iteration.  Both A() and B()
    ' are zero-based vectors so that the coefficent of the constant term in
    ' either polynomial is stored as A(0) or B(0), respectively.
    ' Variables at the beginning and end of one iteration
    Dim Pold As Double
    Dim Qold As Double
    Dim Rold As Double
    Dim Sold As Double
    Dim Pnew As Double
    Dim Qnew As Double
    Dim Rnew As Double
    Dim Snew As Double
    ' Variables to calculate the partial slopes
    Dim Pdel As Double    ' Pdel = P + delP
    Dim RdelP As Double   ' R(P + delP, Q)
    Dim SdelP As Double   ' S(P + delP, Q)
    Dim Qdel As Double    ' Qdel = Q + delQ
    Dim RdelQ As Double   ' R(P, Q + delQ)
    Dim SdelQ As Double   ' S(P, Q + delQ)
    Dim dRdP As Double    ' Partial dR/dP
```

```
Dim dRdQ As Double    ' Partial dR/dQ
Dim dSdQ As Double    ' Partial dS/dQ
Dim BdelP(M) As Double
Dim BdelQ(M) As Double
'////////////////////////////
'// Seed the first iteration.
Pnew = 0.5 * A(1) / Math.Sqrt(A(0))
Qnew = Math.Sqrt(A(0))
B(M) = 0
B(M - 1) = 0
For K As Int32 = M - 2 To 0 Step -1
    B(K) = A(K + 2) - (Pnew * B(K + 1)) - (Qnew * B(K + 2))
Next K
Rnew = A(1) - (Pnew * B(0)) - (Qnew * B(1))
Snew = A(0) - (Qnew * B(0))
' Revise the initial guess using R = S = 0.
Qnew = A(0) / B(0)
Pnew = (A(1) - (Qnew * B(1))) / B(0)
'/////////////
'// Main loop.
Do
    ' Transfer the parameters from "new" to "old".
    Pold = Pnew
    Qold = Qnew
    Rold = Rnew
    Sold = Snew
    ' Vary P and Q.
    Pdel = Pold + 0.000001
    Qdel = Qold + 0.000001
    ' Calculate differentials w.r.t. P.
    BdelP(M) = 0
    BdelP(M - 1) = 0
    For K As Int32 = M - 2 To 0 Step -1
        BdelP(K) = A(K + 2) - (Pdel * B(K + 1)) - (Qold * B(K + 2))
    Next K
    RdelP = A(1) - (Pdel * B(0)) - (Qold * B(1))
    SdelP = A(0) - (Qold * B(0))
    ' Calculate differentials w.r.t. Q.
    BdelQ(M) = 0
    BdelQ(M - 1) = 0
    For K As Int32 = M - 2 To 0 Step -1
        BdelQ(K) = A(K + 2) - (Pold * B(K + 1)) - (Qdel * B(K + 2))
    Next K
    RdelQ = A(1) - (Pold * B(0)) - (Qdel * B(1))
    SdelQ = A(0) - (Qdel * B(0))
    ' Calculate partial derivatives.
    dRdP = (RdelP - Rold) / (Pdel - Pold)
    dRdQ = (RdelQ - Rold) / (Qdel - Qold)
    dSdQ = (SdelQ - Sold) / (Qdel - Qold)
    ' Estimate new P and Q, with convergence factor Alpha.
    Dim Alpha As Double = 0.2
    Pnew = Pold - (Alpha * Rold / dRdP)
    Qnew = Qold - (Alpha * Rold / dRdQ) - (Alpha * Sold / dSdQ)
    ' Calculate new B().
    For K As Int32 = M - 2 To 0 Step -1
        B(K) = A(K + 2) - ((Pnew * B(K + 1)) + (Qnew * B(K + 2)))
    Next K
```

```
        ' Calculate new R and S.
        Rnew = A(1) - (Pnew * B(0)) - (Qnew * B(1))
        Snew = A(0) - (Qnew * B(0))
        ' Determine if it is time to exit.
        If ((Math.Abs(Rnew) <= MaxErr) And _
            (Math.Abs(Snew) <= MaxErr) And _
            (Math.Abs((Pnew - Pold) / Pold) <= MaxErr) And _
            (Math.Abs((Qnew - Qold) / Qold) <= MaxErr)) Then
            Exit Do
        End If
        ' Display the interim results on the screen.
        Form1.labelDisplay.Text = _
            "M = " & Trim(Str(M)) & vbCrLf & _
            "P = " & Trim(Str(Pnew)) & vbCrLf & _
            "Q = " & Trim(Str(Qnew)) & vbCrLf & _
            "R = " & Trim(Str(Rnew)) & vbCrLf & _
            "S = " & Trim(Str(Snew)) & vbCrLf & _
            "dR/dP = " & Trim(Str(dRdP)) & vbCrLf & _
            "dR/dQ = " & Trim(Str(dRdQ)) & vbCrLf & _
            "dS/dQ = " & Trim(Str(dSdQ))
        Form1.labelDisplay.Refresh()
        ' Msgbox("OK to proceed?")
        Application.DoEvents()
    Loop
    P = Pnew
    Q = Qnew
End Sub

Public Sub FactorAllQuadratics(ByVal M As Int32, _
     ByVal A() As Double, _
     ByVal MaxErr As Double)
    ' This subroutine factors a polynomial of even order M, whose coefficients
    ' are passed in as vector A(), into all of its component quadratic
    ' equations.  There will be M/2 quadratic equations.  They are stored as
    ' the first M/2 rows of array Quad(,1).  The quadratic equation represented
    ' by row i is x^2 + Quad(i,1)x + Quad(i,0).
    Dim P As Double
    Dim Q As Double
    Dim B(100) As Double
    '//////////////////////////////
    '// Check for an odd order of M.
    If (Math.Abs(2 * Math.Round((M + 0.1) / 2) - M) > 0.1) Then
        MsgBox("FactorAllQuadratics received an odd-order polynomial")
        Exit Sub
    End If
    '//////////////////////////////////////////
    '// Check for the degenerate case of M = 2.
    If (M = 2) Then
        Quad(1, 1) = A(1)
        Quad(1, 0) = A(0)
        Exit Sub
    End If
    '////////////////////////////////////////////////////////
    '// Main loop to step through the orders of the polynomial.
    Dim II As Int32
    For I As Int32 = 1 To CInt(M / 2) - 1 Step 1
        II = M + 2 - (2 * I)
```

```
        If (II <= 19) Then
            FactorOneQuadratic(II, A, P, Q, B, 0.000000000001)
        Else
            FactorOneQuadratic(II, A, P, Q, B, 0.000000001) ' 1E-9
        End If
        ' Save the quadratic coefficients in Q(M/2,).
        Quad(I, 1) = P
        Quad(I, 0) = Q
        ' Update A() with the reduced polynomial.
        For J As Int32 = 0 To II Step 1
            A(J) = B(J)
        Next J
        ' Display interim results on the screen.
        Form1.labelDisplay.Text = "Quadratic equation is:" & vbCrLf & _
            "P = " & Trim(Str(P)) & vbCrLf & _
            "Q = " & Trim(Str(Q)) & vbCrLf & _
            "Reduced equation is:" & vbCrLf
        For J As Int32 = 1 To II - 2 Step 1
            Form1.labelDisplay.Text = Form1.labelDisplay.Text & _
                "Power " & Trim(Str(I)) & ": coefficient = " & _
                 Trim(Str(B(J))) & vbCrLf
        Next J
        Form1.labelDisplay.Text = Form1.labelDisplay.Text & vbCrLf
        Form1.labelDisplay.Refresh()
        ' MsgBox("OK to proceed")
    Next I
    '//////////////////////////////////////
    '// Save the final remaining quadratic.
    Quad(CInt(M / 2), 1) = B(1)
    Quad(CInt(M / 2), 0) = B(0)
End Sub

Public Sub ComplexAdd( _
    ByVal A As Double, ByVal B As Double, _
    ByVal C As Double, ByVal D As Double, _
    ByRef ResultRe As Double, ByRef ResultIm As Double)
    ' This subroutine adds two complex numbers.
    ' (A + jB) + (C + jD) = ResultRe + jResultIm
    ResultRe = A + C
    ResultIm = B + D
End Sub

Public Sub ComplexMul( _
    ByVal A As Double, ByVal B As Double, _
    ByVal C As Double, ByVal D As Double, _
    ByRef ResultRe As Double, ByRef ResultIm As Double)
    ' This subroutine mulitplies two complex numbers.
    ' (A + jB) x (C + jD) = ResultRe + jResultIm
    ResultRe = (A * C) - (B * D)
    ResultIm = (A * D) + (B * C)
End Sub

Public Sub ComplexSub( _
    ByVal A As Double, ByVal B As Double, _
    ByVal C As Double, ByVal D As Double, _
    ByRef ResultRe As Double, ByRef ResultIm As Double)
    ' This subroutine subtracts two complex numbers.
    ' (A + jB) - (C + jD) = ResultRe + jResultIm
```

```
        ResultRe = A - C
        ResultIm = B - D
    End Sub

    Public Sub ComplexDiv( _
        ByVal A As Double, ByVal B As Double, _
        ByVal C As Double, ByVal D As Double, _
        ByRef ResultRe As Double, ByRef ResultIm As Double)
        ' This subroutine divides two complex numbers.
        ' (A + jB) / (C + jD) = ResultRe + jResultIm
        Dim Denominator As Double
        Denominator = (C * C) + (D * D)
        ResultRe = ((A * C) + (B * D)) / Denominator
        ResultIm = ((B * C) - (A * D)) / Denominator
    End Sub

    Public Sub ComplexPwr( _
        ByVal A As Double, ByVal B As Double, _
        ByVal N As Int32, _
        ByRef ResultRe As Double, ByRef ResultIm As Double)
        ' This subroutine takes an integer power of a complex number
        ' (A + jB)^N = ResultRe + jResultIm
        If (N = 0) Then
            ResultRe = 1
            ResultIm = 0
            Exit Sub
        End If
        If (N = 1) Then
            ResultRe = A
            ResultIm = B
            Exit Sub
        End If
        ResultRe = A
        ResultIm = B
        For I As Int32 = 1 To N - 1 Step 1
            ComplexMul(A, B, ResultRe, ResultIm, ResultRe, ResultIm)
        Next I
    End Sub

End Module
```

```
Option Strict Off
Option Explicit On

Public Module Plotting

    ' Module Plotting produces waveforms with respect to normalized time.

    '////////////////////////////////////////////////////////////////
    '// Description of principal subroutines:

    ' LoadSolution()
    ' Sub LoadSolution() searches through the Excel spreadsheet for the
    ' solution for the N-th order network.  It retrieves the solution
    ' and places it into the variables SolutionExp(), SolutionW(),
    ' SolutionCosRe(,) and SolutionSinRe(,).

    ' AddPlotColumns()
    ' Sub AddPlotColumns() adds the necessary columns to the spreadsheet
    ' to plot the N loop currents and the total current.  If this is
    ' the first call, which is detected by ColNum = 1, then the routine
    ' adds the time column.

    Public Sub LoadSolution( _
        ByVal N As Int32, _
        ByRef SolutionExp() As Double, _
        ByRef SolutionW() As Double, _
        ByRef SolutionCosRe(,) As Double, _
        ByRef SolutionSinRe(,) As Double, _
        ByRef objExcelWS As Microsoft.Office.Interop.Excel.Worksheet, _
        ByRef RowNum As Int32)
        Dim TempString As String
        Dim RowIndex As Int32 = -1
        For Isearch As Int32 = 1 To RowNum Step 1
            With objExcelWS
                If (Not IsNothing(objExcelWS.Cells(Isearch, 1).Value)) Then
                    If (Not IsNumeric(objExcelWS.Cells(Isearch, 1).Value)) Then
                        TempString = .Cells(Isearch, 1).Value.ToString
                        If (TempString = "Complete real matrix for " & _
                            Trim(Str(N)) & " stages:") Then
                            RowIndex = Isearch
                            Exit For
                        End If
                    End If
                End If
            End With
        Next Isearch
        If (RowIndex < 0) Then
            MsgBox("LoadSolution could not find the solution for order " & Trim(Str(N)))
            Exit Sub
        End If
        For Iroot As Int32 = 1 To N Step 1
            With objExcelWS
                TempString = .Cells(RowIndex + 2, 1 + Iroot).Value.ToString
                SolutionExp(Iroot) = Val(TempString)
                TempString = .Cells(RowIndex + 3, 1 + Iroot).Value.ToString
                SolutionW(Iroot) = Val(TempString)
                For Irow As Int32 = 1 To N Step 1
                    Dim Istage As Int32 = N + 1 - Irow
```

```
                TempString = .Cells(RowIndex + 4 + Irow, 1 + Iroot) _
                    .Value.ToString
                SolutionCosRe(Istage, Iroot) = Val(TempString)
                TempString = .Cells(RowIndex + 5 + N + Irow, 1 + Iroot) _
                    .Value.ToString
                SolutionSinRe(Istage, Iroot) = Val(TempString)
            Next Irow
        End With
    Next Iroot
End Sub

Public Sub AddPlotColumns( _
    ByVal N As Int32, _
    ByVal SolutionExp() As Double, _
    ByVal SolutionW() As Double, _
    ByVal SolutionCosRe(,) As Double, _
    ByVal SolutionSinRe(,) As Double, _
    ByRef objExcelWS As Microsoft.Office.Interop.Excel.Worksheet, _
    ByRef RowNum As Int32, ByRef ColNum As Int32)
    Dim T As Double
    Dim CurrentSum As Double
    Dim TotalSum As Double
    '////////////////////////////////
    '// Display message on the screen.
    Form1.labelDisplay.Text = "Plotting currents for order " & Trim(Str(N))
    Form1.labelDisplay.Refresh()
    '////////////////////////////////////////////
    '// If ColNum = 1, then add a column for time.
    '//
    '// StartTime = 0
    '// DelTime = 0.01
    '// EndTime = 25
    '//
    If (ColNum = 1) Then
        With objExcelWS
            .Cells(RowNum + 1, 1) = "Time"
            .Cells(RowNum + 2, 1) = "(s)"
            For Itime As Int32 = 0 To 2500 Step 1
                .Cells(RowNum + 3 + Itime, 1) = Trim(Str(Itime * 0.01))
            Next Itime
        End With
        ColNum = 2
    End If
    '////////////////////////////////////////////
    '// Retrieve the solution from the spreadsheet.
    LoadSolution(N, SolutionExp, SolutionW, _
        SolutionCosRe, SolutionSinRe, objExcelWS, RowNum)
    '//////////////////////
    '// Write the solution.
    With objExcelWS
        .Cells(RowNum, ColNum) = "Currents for " & _
            Trim(Str(N)) & "-stage -->"
        For Istage As Int32 = 1 To N Step 1
            .Cells(RowNum + 1, ColNum + Istage - 1) = _
                "I" & Trim(Str(Istage)) & "(t)"
            .Cells(RowNum + 2, ColNum + Istage - 1) = "(A)"
        Next Istage
        .Cells(RowNum + 1, ColNum + N) = "Itotal(t)"
```

```
            .Cells(RowNum + 2, ColNum + N) = "(A)"
            For Itime As Int32 = 0 To 2500 Step 1
                T = Itime * 0.01
                TotalSum = 0
                For Istage As Int32 = 1 To N Step 1
                    CurrentSum = 0
                    For Iroot As Int32 = 1 To N Step 1
                        CurrentSum = CurrentSum + _
                            (SolutionCosRe(Istage, Iroot) * _
                            Math.Exp(SolutionExp(Iroot) * T) * _
                            Math.Cos(SolutionW(Iroot) * T))
                        CurrentSum = CurrentSum + _
                            (SolutionSinRe(Istage, Iroot) * _
                            Math.Exp(SolutionExp(Iroot) * T) * _
                            Math.Sin(SolutionW(Iroot) * T))
                    Next Iroot
                    .Cells(RowNum + 3 + Itime, ColNum + Istage - 1) = _
                        Trim(Str(CurrentSum))
                    TotalSum = TotalSum + CurrentSum
                Next Istage
                .Cells(RowNum + 3 + Itime, ColNum + N) = Trim(Str(TotalSum))
            Next Itime
        End With
        '//////////////////////////////////////////////
        '// Update ColNum to the end of the spreadsheet.
        ColNum = ColNum + 1 + N
    End Sub

End Module
```