# Solving Linear Algebraic and Differential Equations with L-Systems.

Pavol Federl
University of Calgary

Przemyslaw Prusinkiewicz
University of Calgary

## 1. Introduction

In the previous note it was shown how L-Systems can be used to numerically solve systems of partial differential equations, for a constant or growing medium, and the method was applied to computer graphics purposes. The L-System from the previous section employed a forward Euler method for finite differencing. Although simple to implement, the forward Euler method is in many case inadequate, for example when the equations are stiff. In this note we show how an implicit method for solving differential equations can be implemented within the framework of L-Systems. At the heart of this method lies a technique for solving systems of banded linear equations. To present this method, we use analogy between the processes involved in diffusion and the behavior of electric circuits.

We begin the discussion by describing the electric circuit in Section 2, for which we wish to find the voltages and currents. Then we proceed to develop a set of continuous equations describing the relationship of currents and voltages in the circuits. The continuous time in these equations is then discretized through a Crank-Nicholson implicit finite differencing scheme, described in Section 3. The resulting discretized equations form a set of linear equations, describing the changing state of voltages and currents during a time step. Such a set of equations needs to be then solved at each time step. The set of linear equations can be represented by a 5-diagonal coefficient matrix. In Section 4 we show how such a banded system of linear equations can be effectively solved using L-Systems. In Section 5 we combine the material presented in Sections 3 and 4 into a complete L-System, and in Section 6 we discuss the results.

## 2. The circuit

The circuit we wish to analyze consists of $n$ circuit segments (Figure 2) connected in series (Figure 1). Each segment contains 3 resistors (horizontal, vertical and parallel) and a capacitor. The resistances for the $k$-th segment are labeled $R_k^H$, $R_k^V$ and $R_k^P$, and the capacitance is labeled $C_k$. The voltage $v_k(t)$ on the capacitor, or in short $v_k$, is a function of time. The currents passing through the horizontal, vertical and parallel resistors are $i_k(t)$, $i_k^V(t)$ and $i_k^P(t)$, respectively, while the current passing through the capacitor is $i_k^C(t)$. All 4 currents are also functions of time, and will be referred to using a short-hand notation as $i_k$, $i_k^V$, $i_k^H$ and $i_k^C$, respectively.

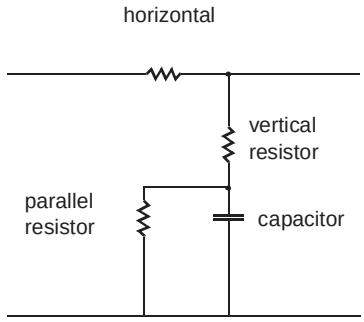



Figure 1: A circuit segment.

At the right end of the circuit, the $n$-th segment is left open. On the left end, the initial segment is connected to a constant voltage source (with voltage $V_S$) and a resistor (with resistance $R_S$).

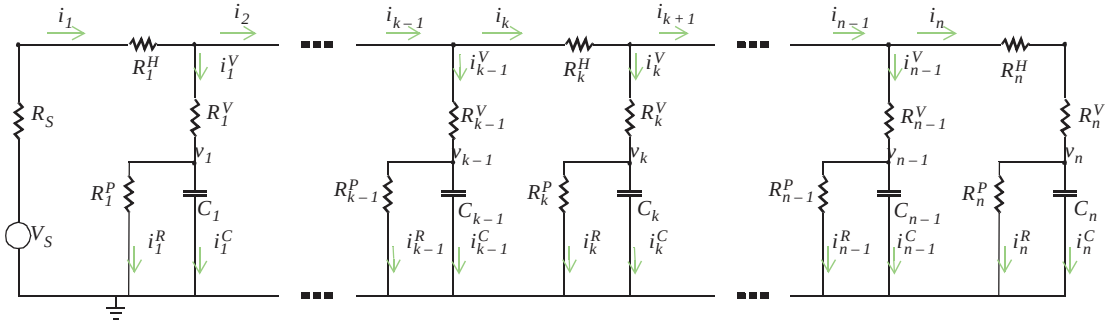



Figure 2: The overall circuit assembled from $n$ segments connected in series.

## 2.1. Continuous equations

In order to analyze the relationship between the currents and voltages in the whole circuit, we consider the currents and voltages from the perspective of a single segment. There are 3 cases to consider: the general case and 2 boundary conditions. In the general case a segment is connect to other segments both on its left side and on its right side. In the boundary cases, a segment is only connected to one other segment. Since the voltage source and the initial resistor are present only on the left side of the circuit, the equations for the left and right boundary cases are derived separately. The initial conditions for the circuit are trivial, i.e. $i_k(0) = v_k(0) = 0$ for all $k$.

### The general case:

The general case applies to a segment $k$ when $k \in [2, n-1]$. In such cases the segment $k$ is connected to a segment $k-1$ on the left, and to segment $k+1$ on the right. We can express the voltage $v_k$ and current $i_k$ in terms of currents $i_{k-1}$ and $i_{k+1}$, and the voltage $v_{k-1}$ as follows. The relationship between voltages $v_{k-1}$ and $v_k$ is established using Kirchoff's voltage law (KVL), i.e.:

Eq.1
$$v_k = v_{k-1} + R_{k-1}^V i_{k-1}^V - R_k^H i_k - R_k^V i_k^V .$$

From Kirchoff's current law (KCL) we know that $i_{k-1}^V = i_{k-1} - i_k$ and $i_k^V = i_k - i_{k+1}$. Substituting these into Eq. 1 results in:

$$v_k = v_{k-1} + R_{k-1}^V (i_{k-1} - i_k) - R_k^H i_k - R_k^V (i_k - i_{k+1}) \quad , \text{or}$$

$$v_k = v_{k-1} + R_{k-1}^V i_{k-1} - R_{k-1}^V i_k - R_k^H i_k - R_k^V i_k + R_k^V i_{k+1} , \text{or}$$

Eq.2
$$R_{k-1}^V i_{k-1} + v_{k-1} - (R_{k-1}^V + R_k^H + R_k^V) i_k - v_k + R_k^V i_{k+1} = 0 \quad .$$

The current passing through the capacitor is defined as a time derivative of the voltage on the capacitor, i.e.:

Eq.3
$$C_k \frac{dv_k}{dt} = i_k^C .$$

From KVL we know that $v_k - R_k^P i_k^R = 0$ and from KCL we know that $i_k^R = i_k^V - i_k^C$ and therefore $i_k^R = i_k - i_{k+1} - i_k^C$. Substituting these into Eq. 3 yields:

Eq.4
$$R_k^P C_k \frac{dv_k}{dt} = R_k^P (i_k - i_{k+1}) - v_k \quad .$$

Equations 2 and 4 provide an implicit relationship between the voltages and currents of a segment $k$, and the voltages and currents of its immediate neighbors: segments $k-1$ and $k+1$.

The right boundary case:

The equations for the right boundary case can be derived by considering the current $i_n$ and voltage $v_n$ for segment $n$. Since the circuit to the right of segment $n$ is open, the equations for the right boundary case can be derived directly from the general case by setting $k = n$ and $i_{n+1} = 0$. Eq. 2 then becomes:

Eq.5 $$-R^V_{n-1} i_{n-1} - v_{n-1} + (R^V_{n-1} + R^H_n + R^V_n) i_n + v_n = 0,$$

and Eq. 4 becomes:

Eq.6 $$R^P_n C_n \frac{dv_n}{dt} = R^P_n i_n - v_n.$$

The left boundary case:

The voltage $v_1$ can be derived again from KVL:

$$v_1 = V_S - (R_S + R^H_1) i_1 - R^V_1 i^V_1, \text{ or}$$

$$v_1 = V_S - (R_S + R^H_1 + R^V_1) i_1 + R^V_1 i_2, \text{ or}$$

Eq.7 $$(R_S + R^H_1 + R^V_1) i_1 + v_1 - R^V_1 i_2 = V_S.$$

By following the same process for deriving Eq. 4, we arrive to the second equation for the first circuit segment:

Eq.8 $$R^P_1 C_1 \frac{dv_k}{dt} = R^P_1 (i_1 - i_2) - v_1.$$

## 3. Discretization of time

For a circuit with $n$ segments, $2n$ coupled equations describe the relationships between currents and voltages. Of the $2n$ equations, $2$ correspond to the left boundary case (Equations 7 and 8), $2(n-2)$ equations come from the general case (Equations 2 and 4), and $2$ equations correspond to the right boundary case (Equations 5 and 6). One half of these equations is a set of regular algebraic equations, while the other half is a set of first order differential equations. An analytical solution to such a system of equations is unfeasible even for moderate values of $n$, and thus numerical solution becomes a necessity.

We find a numerical solution to these equations through finite differencing. To this end, we discretize the time domain into discrete time steps, each of size $\Delta t$. The solution of current $i_k(t)$ is approximated by $I^m_k$, where the integer $m$ represents the discretized time $t$, i.e. $t = m\Delta t$. Similarly, the solution for voltage $v_k(t)$ is approximated by $V^m_k$. For example, the initial condition corresponding to $i_k(0) = v_k(0) = 0$ is expressed by setting

Eq.9 $$I^0_k = V^0_k = 0 \text{ for all } k.$$

The $n$ algebraic equations (7, 2 and 5) provide a relationship between the currents and voltages at a specific time $t$. These equations are therefore discretized by applying the following substitutions:

Eq.10 $$i_p \to I^m_p \text{ and } v_p \to V^m_p.$$

The set of $n$ first order differential equations are discretized using the Crank-Nicholson finite differencing scheme, using the following substitutions:

Eq.11 $$i_p \to \frac{I^m_p + I^{m-1}_p}{2}, \; v_p \to \frac{V^m_p + V^{m-1}_p}{2} \text{ and } \frac{dv_p}{dt} \to \frac{V^m_p - V^{m-1}_p}{\Delta t}.$$

The discretized equations 7, 8, 2, 4, 5 and 6 take on the following form:

Eq.12 $$(R_S + R_1^H + R_1^V)I_1^m + V_1^m - R_1^V I_2^m = V_S,$$

Eq.13 $$-\Delta t R_1^P I_1^m + (2R_1^P C_1 + \Delta t)V_1^m + \Delta t R_1^P I_2^m = \Delta t R_1^P I_1^{m-1} - \Delta t R_1^P I_2^{m-1} + (2R_1^P C_1 - \Delta t)V_1^{m-1},$$

Eq.14 $$R_{k-1}^V I_{k-1}^m + V_{k-1}^m - (R_{k-1}^V + R_k^H + R_k^V)I_k^m - V_k^m + R_k^V I_{k+1}^m = 0,$$

Eq.15 $$-\Delta t R_k^P I_k^m + (2R_k^P C_k + \Delta t)V_k^m + \Delta t R_k^P I_{k+1}^m = (2R_k^P C_k - \Delta t)V_k^{m-1} + \Delta t R_k^P I_k^{m-1} - \Delta t R_k^P I_{k+1}^{m-1},$$

Eq.16 $$R_{n-1}^V I_{n-1}^m + V_{n-1}^m - (R_{n-1}^V + R_n^H + R_n^V)I_n^m - V_n^m = 0, \text{ and}$$

Eq.17 $$-\Delta t R_n^P I_n^m + (2R_n^P C_n + \Delta t)V_n^m = (2R_n^P C_n - \Delta t)V_n^{m-1} + \Delta t R_n^P I_n^{m-1}.$$

The above equations form a set of linear equations, where the unknown values are the approximations of currents and voltages for a given time, i.e. $I_1^m$, $V_1^m$, $I_2^m$, $V_2^m$ ... $I_n^m$, $V_n^m$. These unknown values are implicitly defined from the old values $I_1^{m-1}$, $V_1^{m-1}$, $I_2^{m-1}$, $V_2^{m-1}$ ... $I_n^{m-1}$, $V_n^{-1}$, and from additional constants. The process of finding a numerical solution to the circuit problem consists of starting with the initial condition (a set of known values for time $m = 0$), and then iteratively finding the unknowns for the next time step.

## 3.1. 5-diagonal system of linear equations

The system of linear Equations 12-17 can be written in the matrix form as:

Eq.18 $$Ax = b,$$

where $x$ is the column vector representing the solution:

Eq.19 $$x^T = \left[I_1^m\ V_1^m\ I_2^m\ V_2^m\ I_3^m\ V_3^m\ \ldots\ I_n^m\ V_n^m\right],$$

$b$ is a column vector containing the right hand sides of the equations:

Eq.20 $$b^T = \left[B_1\ B_2\ \ldots\ B_{2n-1}\ B_{2n}\right]$$

and $A$ is a $2n$ by $2n$ matrix containing the coefficients of the unknowns:

Eq.21 $$A = \begin{bmatrix} A_{1,1} & A_{1,2} & & A_{1,2n} \\ A_{2,1} & A_{2,2} & \ldots & A_{2,2n} \\ & \ldots & & \ldots \\ A_{2n,1} & A_{2n,2} & \ldots & A_{2n,n} \end{bmatrix}.$$

For even values of $j$, the entries $A_{i,j}$ represents the coefficient for the unknowns $I_{i/2}^m$, while for odd values of $j$, the entries $A_{i,j}$ correspond to the coefficients of $V_{\lfloor i/2 \rfloor}^m$. A quick inspection of Equations 12-17 reveals that the coefficient matrix $A$ is 5-diagonal, i.e. that for a given row $i$, only entries $A_{i,i-2}$, $A_{i,i-1}$, $A_{i,i}$, $A_{i,i+1}$ and $A_{i,i+2}$ can be non-zero. The fact that matrix is 5-diagonal is important in that it allows for efficient solution to the system of linear equations. Specifically, such a system can be solved in linear time.

The techniques for solving a set of LEs can be divided into two main categories: direct and iterative methods. Direct methods solve the equations by algebraic manipulations, while iterative methods solve the equations by improving an existing solution in successive iterations. Gaussian elimination falls into the category of direct solution techniques and runs in average time of $O(n^3)$. It solves the system of LEs by successively simplifying the original system, which is achieved in two phases. In the first phase, the LEs are adjusted so that all non-zero coefficients below the diagonal are eliminated. In the second phase, the entries above the diagonal are eliminated.

The Gaussian elimination algorithm can be made more efficient when the coefficient matrix of the linear system is banded. In a banded matrix all nonzero components tend to group around the diagonal. The number describing how

well the given matrix is banded is called the bandwidth of the matrix, and it is the width of a diagonal band (or strip) which completely encompasses all non-zero elements of a matrix. A 5-diagonal matrix has a bandwidth equal to 5. If the bandwidth of a given matrix is $m$, then the Gaussian elimination algorithm can be modified so that the running time is $O(m^2 n)$. This is achieved by modifying the original Gaussian elimination algorithm to perform row subtractions only in the areas where there are non-zero entries. The running time of Gaussian elimination on a 5-diagonal system of LEs is therefore $O(n)$.

## 4. Solving 5-diagonal systems of linear equations using L-Systems

Here we show how L-Systems can be used to solve a system of $n$ linear equations which in matrix form can be written as $Ax = b$, and when the coefficient matrix $A$ is a 5 diagonal matrix of the form:

$$A = \begin{bmatrix} a_{1,3} & a_{1,4} & a_{1,5} & 0 & 0 & 0 \\ a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} & 0 & 0 \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} & 0 \\ 0 & a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\vdots \qquad \cdots$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} a_{n-3,1} & a_{n-3,2} & a_{n-3,3} & a_{n-3,4} & a_{n-3,5} & 0 \\ 0 & a_{n-2,1} & a_{n-2,2} & a_{n-2,3} & a_{n-2,4} & a_{n-2,5} \\ 0 & 0 & a_{n-1,1} & a_{n-1,2} & a_{n-1,3} & a_{n-1,4} \\ 0 & 0 & 0 & a_{n,1} & a_{n,2} & a_{n,3} \end{bmatrix}$$

The column vector $x = [x_1 \ldots x_n]^T$ represents the $n$ unknowns, and the column vector $b = [b_1 \ldots b_n]^T$ represents the right hand sides of the equations. To represent the system of equations using an L-System string, we use a string of $n$ modules `M`. Each module `M` has a set of values associated with it, representing all non-zero coefficients of a single row of the matrix, plus the corresponding entry of the solution vector $b$. The values are grouped into a single parameter of type `struct Row`:

```
struct Row
{
     double a1, a2, a3, a4, a5, rhs;
     Row () { a1 = a2 = a3 = a4 = a5 = rhs = 0.0; }
     Row (double p1, double p2, double p3, double p4, double p5, double pb)
            { a1 = p1; a2 = p2; a3 = p3; a4 = p4; a5 = p5; rhs = pb;}
};
```

The first module `M` represents the first row of the matrix $A$ and the column vector $b$, the second module `M` represents the second row, etc. For example, consider the following system of 6 linear equations of 6 unknowns:

$$
\begin{aligned}
7x_1 + 8x_2 - 3x_3 &= 10 \\
4x_1 - x_2 + 3x_3 + 4x_4 &= -2 \\
-x_1 + x_3 + x_5 &= 7 \\
x_2 - 2x_3 + x_4 &= 0 \\
x_3 + 2x_4 + 3x_5 + 4x_6 &= 5 \\
2x_2 + 4x_5 + 6x_6 &= 8
\end{aligned}
$$

This system, in matrix form can be written as: $Ax = b$, where

$$
A = \begin{bmatrix} 7 & 8 & -3 & 0 & 0 & 0 \\ 4 & -1 & 3 & 4 & 0 & 0 \\ -1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 & 3 & 4 \\ 0 & 0 & 0 & 2 & 4 & 6 \end{bmatrix}, \; x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} \text{ and } b = \begin{bmatrix} 10 \\ -2 \\ 7 \\ 0 \\ 5 \\ 8 \end{bmatrix}.
$$

The L-System string of modules representing such a system would be:

```
M(Row(0,0,7,8,-3,10)) M(Row(0,4,-1,3,4,-2)) M(Row(-1,0,1,0,1,7)) M(Row(1,-2,1,0,0,0))
M(Row(1,2,3,4,0,5)) M(Row(2,4,6,0,0,8))
```

## 4.1. First phase - elimination below diagonal

As described in the previous section, the solution to the system of linear equation is found by performing a two-phase process. In the first phase, the coefficients below the diagonal are eliminated. This corresponds to re-writing each module `M` of the string corresponding to rows $2...n$, processing the string from left to right. Let us assume the first $k-1$ modules have been already rewritten, i.e. the first $k-1$ rows of $A$ already have $0$ 's below the diagonal:

$$
A = \begin{bmatrix} & & & \cdots & & & \\ \cdots & 0 & a_{k-2,3} & a_{k-2,4} & a_{k-2,5} & 0 & 0 & 0 \\ & 0 & 0 & a_{k-1,3} & a_{k-1,4} & a_{k-1,5} & 0 & 0 \\ & 0 & a_{k,1} & a_{k,2} & a_{k,3} & a_{k,4} & a_{k,5} & 0 \\ & & & \cdots & & & & \end{bmatrix} \cdots
$$

To adjust the row $k$ so that its two non-zero entries below diagonal are eliminated, proper multiples of rows $k-2$ and $k-1$ have to be subtracted from row $k$. This is achieved by simultaneously adjusting the coefficients of row $k$ as follows:

Eq.22

$$\left.\begin{aligned} a_{k,1} &\rightarrow 0 \\ a_{k,2} &\rightarrow 0 \\ a_{k,3} &\rightarrow a_{k,3} - f_1 a_{k-2,4} - f_2 a_{k-1,4} \\ a_{k,4} &\rightarrow a_{k,4} - f_2 a_{k-1,5} \\ a_{k,5} &\rightarrow a_{k,5} \\ b_k &\rightarrow b_k - f_1 b_{k-2} - f_2 b_{k-1} \end{aligned}\right\} \text{ where } \begin{aligned} f_1 &= \frac{a_{k,1}}{a_{k-2,3}} \\ f_2 &= \frac{a_{k-2,3}a_{k,2} - a_{k-2,4}a_{k,1}}{a_{k-2,3}a_{k-1,3}} \end{aligned} .$$

The above substitutions cannot be applied to row $2$, as there is only a single row above it. One solution is to treat row $2$ as a special case. Another solution, which does not require handling of a special case, is to include a phony row $0$, with coefficients $a_{0,3} = 1$ and $a_{0,1} = a_{0,2} = a_{0,4} = a_{0,5} = b_0 = 0$.

The L-System production that performs the pass from left to right - eliminating all entries in the coefficient matrix below the diagonal - is shown below:

```
M(r1) M(r2) << M(r3) :
{
     if (phase == LEFT_TO_RIGHT)
     {
             double f1 = r3.a1 / r1.a3;
             double f2 = (r1.a3 * r3.a2 - r1.a4 * r3.a1) / (r1.a3 * r2.a3);
             produce M(Row(0, 0, r3.a3-f1*r1.a5-f2*r2.a4, r3.a4-f2*r2.a5
                         , r3.a5, r3.rhs-f1*r1.rhs-f2*r2.rhs));
     }
}
```

This production effectively replaces each row with a new row, by subtracting from it the proper multiples of the 2 rows above it. In the end, the replaced row contains 0's to the left of the diagonal coefficient. The production rule uses fresh left context to gain access to the already modified 2 rows above the row to be adjusted. By using fresh left context, this production rule is applied to the modules in a single pass.

## 4.2. Second phase - elimination above diagonal

Once the first phase is completed, the coefficient matrix has the form:

$$A = \left[\begin{array}{|c|c|c|c|c|c|c|}\hline a_{1,3} & a_{1,4} & a_{1,5} & 0 & 0 & 0 & 0 \\ \hline 0 & a_{2,3} & a_{2,4} & a_{2,5} & 0 & 0 & 0 \\ \hline 0 & 0 & a_{3,3} & a_{3,4} & a_{3,5} & 0 & 0 \\ \hline \end{array}\right. \begin{array}{c} \vdots \\ \\ \\ \end{array} \left.\begin{array}{|c|c|}\hline 0 & 0 \\ \hline 0 & 0 \\ \hline 0 & 0 \\ \hline \end{array}\right]$$

$$\cdots$$

$$\left[\begin{array}{|c|c|c|c|c|c|c|}\hline 0 & 0 & 0 & 0 & 0 & a_{7,3} & a_{7,4} \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & a_{n,3} \\ \hline \end{array}\right]$$

To finish the process of finding the solution, the coefficients above the diagonal are eliminated in the second phase. This is achieved by processing the rows from bottom to top, subtracting from each row the appropriate multiples of the two rows below. Assuming that $k$ rows remain to be processed, the $k$-th row is adjusted by subtracting from it multiples of rows $k + 1$ and $k + 2$:

Eq.23

$$\left.\begin{aligned} a_{k,1} &\to 0 \\ a_{k,2} &\to 0 \\ a_{k,3} &\to a_{k,3} \\ a_{k,4} &\to 0 \\ a_{k,5} &\to 0 \\ b_k &\to b_k - f_1 b_{k+1} - f_2 b_{k+2} \end{aligned}\right\} \text{ where } \begin{aligned} f_1 &= \frac{a_{k,4}}{a_{k+1,3}} \\ f_2 &= \frac{a_{k,5}}{a_{k+2,3}} \end{aligned} .$$

Again, the above substitutions cannot be applied to row $n-1$, as it does not have two rows below it. In order to avoid writing an extra rule handling a special case, we add instead a phone row $n+1$ with the same coefficients as row 0, i.e. $a_{n+1,3} = 1$ and $a_{n+1,1} = a_{n+1,2} = a_{n+1,4} = a_{n+1,5} = b_{n+1} = 0$. The production rule which effects the second phase is:

```
M(r1) >> M(r2) M(r3) :
{
     if (phase == RIGHT_TO_LEFT)
     {
             produce M(Row(0, 0, r1.a3, 0, 0
                         , r1.rhs - r1.a4*r2.rhs/r2.a3 - r1.a5*r3.rhs/r3.a3));
     }
}
```

Similar to the production rule used in the left-to-right phase, this right-to-left production rule also uses fresh context. Combined with processing the string of modules from right-to-left, this production rule is applied to the whole string in a single pass. After the second phase is finished, the coefficient matrix has the form:

$$A = \begin{bmatrix} a_{1,3} & 0 & 0 & 0 & 0 & \vdots & 0 & 0 \\ 0 & a_{2,3} & 0 & 0 & 0 & & 0 & 0 \\ 0 & 0 & a_{3,3} & 0 & 0 & & 0 & 0 \\ & & & & & \cdots & & \\ 0 & 0 & 0 & 0 & 0 & & a_{7,3} & 0 \\ 0 & 0 & 0 & 0 & 0 & & 0 & a_{n,3} \end{bmatrix}$$

Since non-zero entries are only on the diagonal, retrieving the solution is trivial, i.e. $x_i = b_i / a_{i,3}$.

# 5. The complete L-System implementation

In Section 3 we have shown how the continuous equations describing the voltages and currents in the circuit have been discretized. The result of such a discretization is a set of linear equations, which must be solved at each time step. Since the set of linear equations is represented by a 5-diagonal coefficient matrix, they can be solved using L-Systems, as demonstrated in Section 4. In this Section we describe a complete L-System implementation of the solution (the complete source is given at the end of this report).

The overall operation of the L-System can be distinguished into 5 distinct phases:

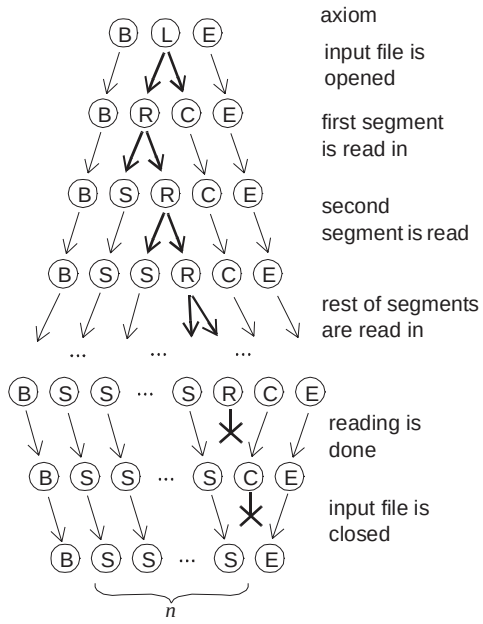
- Phase 0: a data-file describing the circuit is loaded;
- Phase I: the 5 diagonal coefficient matrix and the right hand-side is set up;

- Phase II: the entries below diagonal are eliminated;
- Phase III: the entries above diagonal are eliminated;
- Phase IV: the solution is extracted and simulation time is advanced.

Once the circuit has been loaded from the file, the L-System cycles through phases I-IV. Phase I, II and IV require the string processing to be done from left-to-right, and phase III needs the string to be processed from right-to-left. Since different production rules need to be applied in different phases, a global variable `phase` is used to denote the current phase. The symbolic names of the 4 phases are `SETUP`, `LEFT_TO_RIGHT`, `RIGHT_TO_LEFT` and `COLLECT`. Initially, `phase` is set to `SETUP` (line 53). At the end of each string rewrite, the `phase` is adjusted to reflect the next stage (lines 55-63). Notice that for phase III the string processing is reversed (line 60).

## Phase 0: reading in the data-file

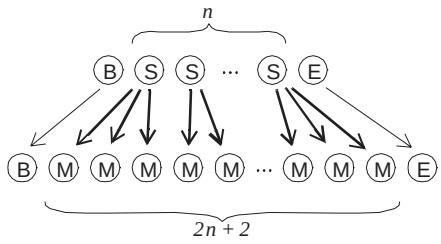
The L-System string is initialized through an axiom (line 65), to contain 3 modules: `B L` and `E`. Modules `B` and `E` denote the beginning and the end of the string, respectively, which are used to determine boundary case conditions. The parameter of module `L` is a string, which specifies the data-file from which the circuit will be loaded. Through decomposition rules (lines 68-98), the data-file is first opened, and then read in segment by segment. At the end of the decomposition, the string has the form "`B S S ... S E`". There is one module S for each circuit segment. The graphical representation of this process is illustrated in the figure on the right.




Each module `S` has a parameter of type struct Segment (defined on lines 17-21). The fields `Rh`, `Rv`, `Rp` and `Cap` are read in from the file, and represent the 3 resistances and a capacitance of the segment. The fields `I` and `V` contain the calculated current and voltage in the segment, and are both initialized to 0 to reflect the initial condition (Eq. 9).

## Phase I: setting up the matrix representation

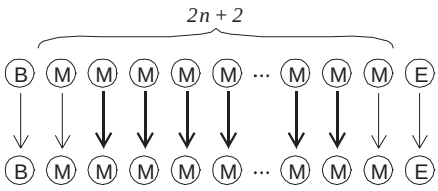
In phase I, the string is rewritten to represent the system of 5-diagonal linear equations, implemented by the productions of group `SETUP` (lines 103-120). This is achieved by replacing each module `S` with 2 modules `M`, where each module `M` represents a row in the coefficient matrix $A$ and the corresponding row of the right hand side column vector $b$, as described in Section 4. If there are $n$ segments in the circuit, there would be $2n + 2$ rows (or modules `M`).




The production rule on lines 103-109 corresponds to the general case, and is applied to all segments that have both neighbors. This rule is derived directly from Eq. 14 and Eq. 15. The production rule on lines 110-115 reflects the left-boundary case, and is only applied to the very first segment (represented by module `S` that immediately follows module `B`). The left boundary case production rule was derived to reflect Eq. 12 and Eq. 13. Finally, the right boundary case (lines 116-120) is applied to the right-most segment, and corresponds to Eq. 16 and Eq. 17. The right-most segment is determined by requiring the right context of the module `S` to be module `E`. Finally, notice how the extra rows are appended at the beginning and at the end of the string (line 111 and line 119).

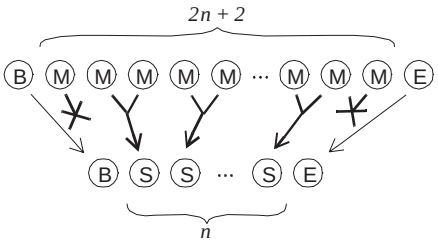
### Phase II and Phase III: solving the system of equations

In phases II and III the string of modules is rewritten to represent a system of equations, where the corresponding coefficient matrix has non-zero entries only on its diagonal (lines 121-132). These two phases have been explained in detail in Section 4.




### Phase IV: extracting the solution

In the last phase, the solution is extracted from the string (lines 133-148). First, the extra rows are eliminated by the production rules on lines 135-140. Then, for every pair of modules `M`, a modules `S` is produced (lines 141-145). The current for the produced module S is calculated from the first `M`, while the voltage is calculated from the second `M`.




### Rendering

The circuit is rendered each time at the end of phase IV, done by the interpretation rules (lines 150-228). All of the results presented in the next section were rendered using this L-System, both the circuit and the graphs.

## 6. Results and conclusions

The output of the L-System program for a simple circuit composed of 4 different segments is shown in Figure 3. At the top of Figure 3 the circuit is rendered. At the bottom of the figure, the graphs of voltages and currents are displayed at 5 different points in time.

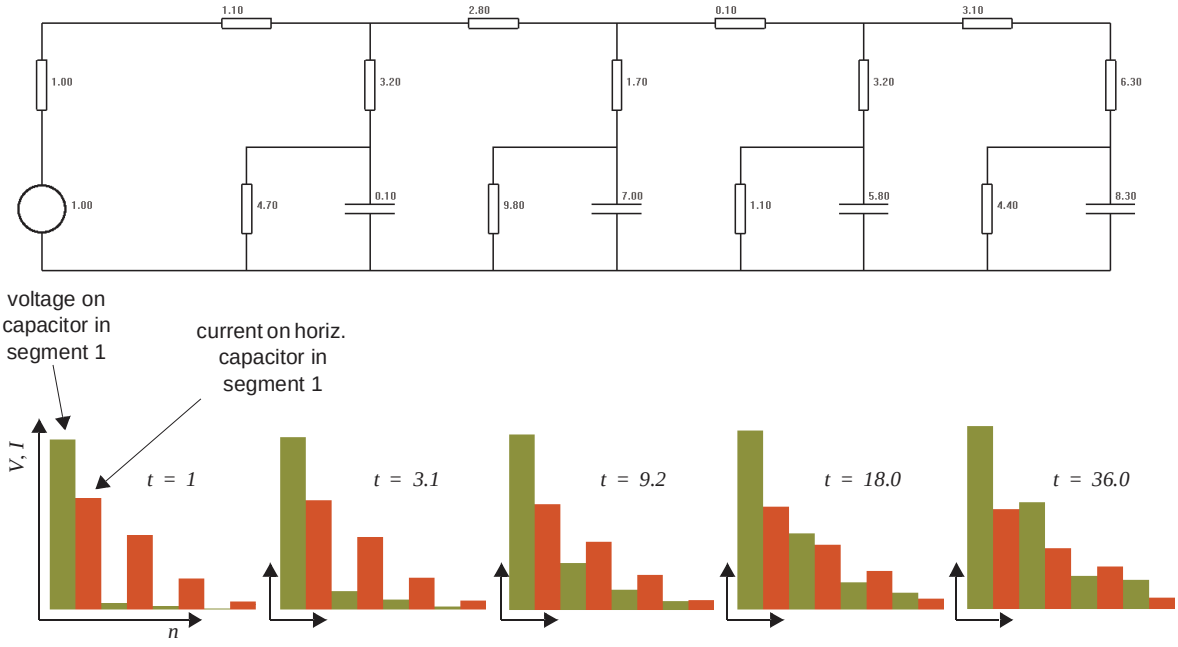



Figure 3: Example I - simple circuit composed of 4 different segments.

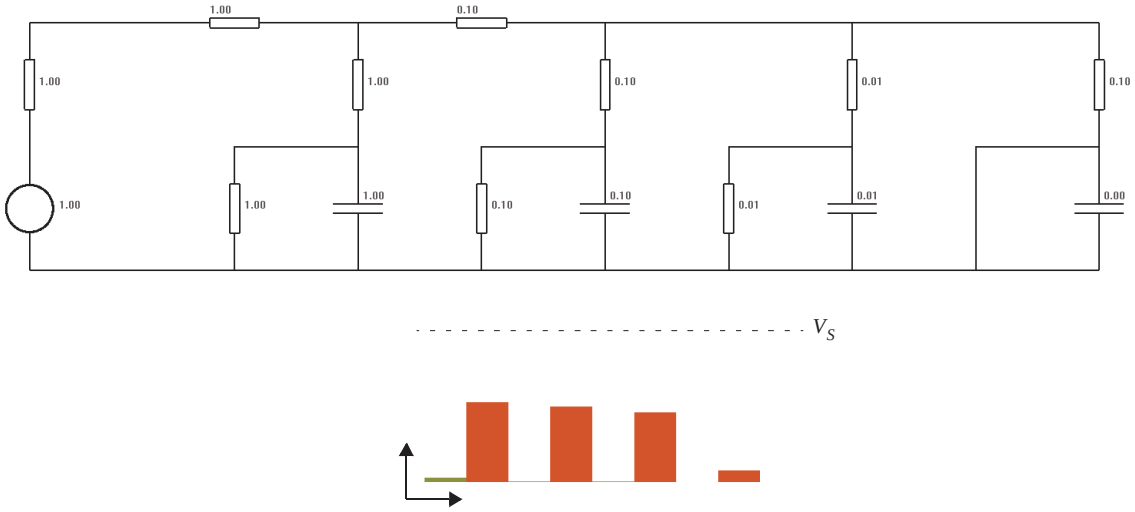



Figure 4: Example II - circuit with some resistances set to 0.

The next example illustrates the robustness of the presented method for solving the circuit analysis problem. The circuit in Figure 4 contains resistances and capacitances differing in magnitudes, some of which are even set to 0. Such a circuit leads to stiff equations, which are impossible to solve using forward integration methods. Our implicit method however, solves the problem successfully.

The circuit in Figure 5 is set up to simulate diffusion. It is composed of 50 identical segments. Because of the presence of the parallel resistors, less and less current propagates to the capacitors toward the right-hand-side of the circuit. As a result, the further the capacitor is from the voltage source, the less it will be charged. This is very similar to diffusion with decay, where the concentration of the chemical decreases as the distance from the source increases.

Finally, we have also simulated a circuit analogous to diffusion without decay, illustrated in Figure 6. This was achieved by setting the relative differences between the resistances of the parallel resistors and the other two types of resistors very high. As a result, given enough time - all capacitors are eventually charged to the same level.

In conclusion, we have successfully demonstrated how L-Systems can be used to solve systems of differential equations, while employing implicit finite differencing scheme. The core of this approach lies in the method of solving banded systems of linear equations using L-Systems, which we described in detail. Using fresh contexts, we were able to implement the solution to solve the equations in only two passes. The resulting L-System performs almost as fast as a straight C++ implementation, proving that L-Systems are a viable mechanism for solving systems of differential equations in modeling problems.

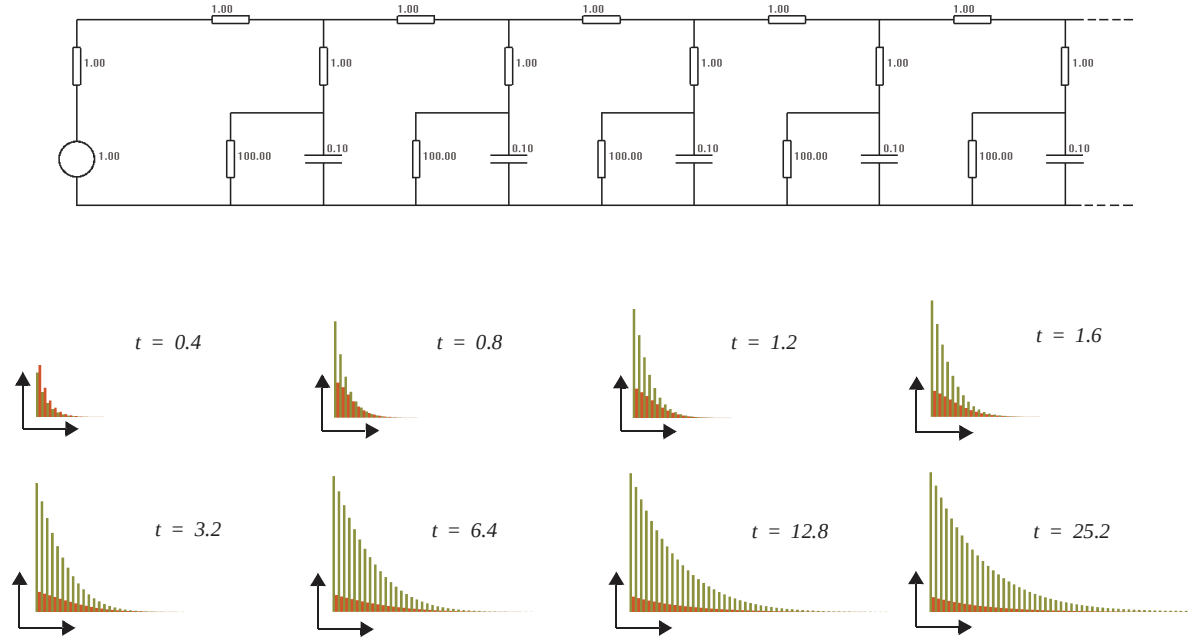



Figure 5: Example III - simulated diffusion, with decay.

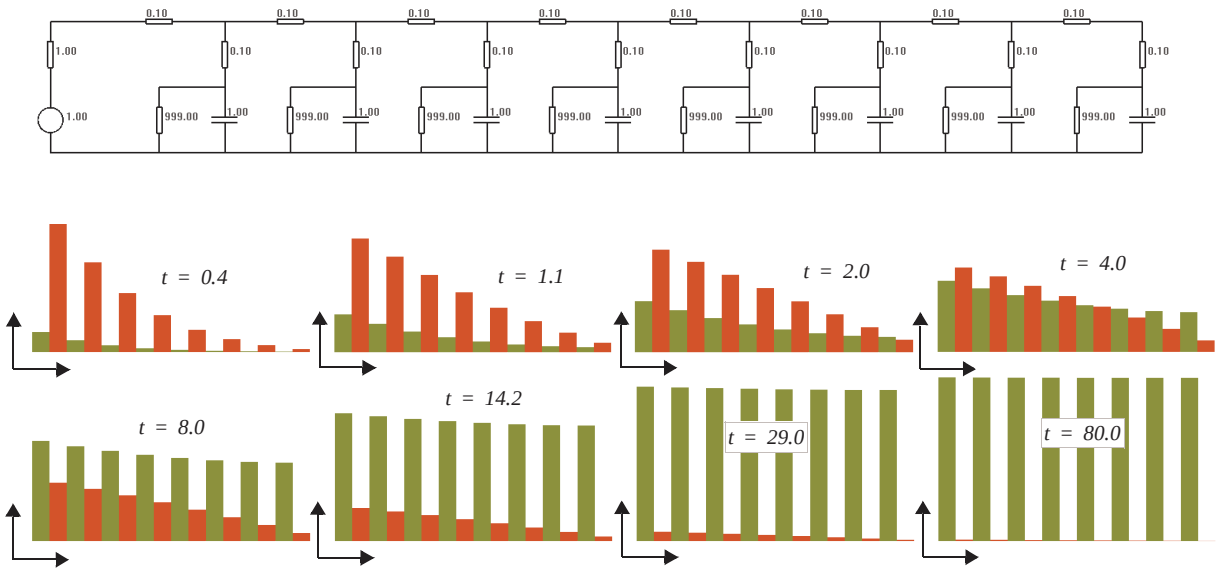



Figure 6: Example III - simulated diffusion without decay.

# Appendix A: The complete L+C source code

```
#include <cmath>
#include <cstdlib>
#include <lpfgall.h>
#include <cstdio>
#include <cstdlib>
#include <string>
#include <cassert>
#include <stdarg.h>

using std::string;

const string fname = "circuit-1.dat"; FILE * fp;
double dt, curr_time, Vs, Rs; // time step, curr. time, Rs & Vs
float x, vscale;// used for rendering
bool draw_circuit;// whether to draw circuit

struct Segment
{   double Rh, Rv, Rp, Cap;// the resistances and the capacitance of a single segment
    double I, V;// current and voltage
    Segment () { Rh = Rv = Rp = Cap = I = V = 0.0; }
};

struct Row
{   double a1, a2, a3, a4, a5, rhs;
    Segment seg;
    Row () { a1 = a2 = a3 = a4 = a5 = rhs = 0.0; }
    Row (double pa1, double pa2, double pa3, double pa4, double pa5, double prhs, Segment & pseg)
       { a1 = pa1; a2 = pa2; a3 = pa3; a4 = pa4; a5 = pa5; rhs = prhs; seg = pseg;}
};

module B();                              // marks the beginning of the string
module E();                              // marks the end of the string
module L(string);                        // module that will load the file
module C();                              // closes the file
module R(long);                          // reads the file
module S(Segment);                       // contains the information about the segment
module M(Row);                           // represents one row of coeff. matrix & RHS
module Capacitor(double, double, double);// renders a capacitor
module ResistorV(double, double, double);// renders a vertical resistor
module ResistorH(double, double, double);// renders a horizontal resistor
module Emf(double, double, double);      // renders EMF
module Rectangle(double, double, double, double); // draw empty rectangle
module RectangleF(double, double, double, double); // draw filled rectangle
module LabS(double, double, string);     // draw a string

// phases of computation
#define SETUP 1
#define LEFT_TO_RIGHT 2
#define RIGHT_TO_LEFT 3
#define COLLECT 4
int phase;

Start: {phase = SETUP;Forward(); }
StartEach: {UseGroup (phase); }
EndEach:
{   switch (phase)
    {
    case SETUP:          phase = LEFT_TO_RIGHT;  Forward();   break;
    case LEFT_TO_RIGHT:  phase = RIGHT_TO_LEFT;  Backward();  break;
    case RIGHT_TO_LEFT:  phase = COLLECT;        Forward();   break;
    case COLLECT:        phase = SETUP;          Forward();   break;
    }
}

   Axiom: B() L(fname) E();
// ============================================================
```

```
67
68    decomposition:
69    maximum depth: 1000;
70 // ============================================================
71 L(fname) : // open the file for reading
72 {   fp = fopen (fname . c_str (), "r");
73     bool error = (fp == NULL);
74     long nseg;
75     error = error || (4 != fscanf (fp, "%lf %ld %lf %lf", & dt, & nseg, & Vs, & Rs));
76     if (error){
77        Printf ("Cannot open/read file %s.\n", fname . c_str ());
78        produce ;
79     }
80     draw_circuit = nseg < 10;
81     vscale = nseg;
82     curr_time = 0.0;
83     produce R(nseg) C();
84 }
85 C() : // Close the file
86 {   fclose (fp);
87     produce ;
88 }
89 R(n) : // Read another segment from the file
90 {   if (n == 0) produce ;
91     Segment s;
92     if (4 != fscanf (fp, "%lf %lf %lf %lf", & s.Rh, & s.Rv, & s.Rp, & s.Cap))
93     {
94        Printf ("Cannot read segment.\n");
95        produce ;
96     }
97     produce S(s) R(n-1);
98 }
99
100    production:
101    derivation length: 4;
102 // ============================================================
103    group SETUP:
104 // ----------------------------------------
105 S(sL) < S(sC) > S(sR) : // general case
106 {   produce M (Row (-sL.Rv, -1, sL.Rv+sC.Rh+sC.Rv, 1, -sC.Rv, 0, sC))
107        M (Row (0, -dt*sC.Rp, 2*sC.Rp*sC.Cap+dt, dt*sC.Rp, 0
108               , (2*sC.Rp*sC.Cap-dt)*sC.V+dt*sC.Rp*sC.I-dt*sC.Rp*sR.I, sC));
109 }
110 B() < S(sC) > S(sR) : // left boundary case
111 {   produce M (Row (0,0,1,0,0,0,sC))
112        M (Row (0, 0, Rs+sC.Rh+sC.Rv, 1,-sC.Rv, Vs, sC))
113        M (Row (0, -dt*sC.Rp, 2*sC.Rp*sC.Cap+dt, dt*sC.Rp, 0
114               , (2*sC.Rp*sC.Cap-dt)*sC.V+dt*sC.Rp*sC.I-dt*sC.Rp*sR.I, sC));
115 }
116 S(sL) < S(sC) > E() : // right boundary case
117 {   produce M (Row (-sL.Rv, -1, sL.Rv+sC.Rh+sC.Rv, 1, 0, 0, sC))
118        M (Row (0, -dt*sC.Rp, 2*sC.Rp*sC.Cap+dt, 0, 0, (2*sC.Rp*sC.Cap-dt)*sC.V+dt*sC.Rp*sC.I, sC))
119        M (Row (0,0,1,0,0,0,sC));
120 }
121    group LEFT_TO_RIGHT:
122 // ----------------------------------------
123 M(r1) M(r2) << M(r3) :
124 {   double k1 = r3.a1 / r1.a3;
125     double k2 = (r1.a3 * r3.a2 - r1.a4 * r3.a1) / (r1.a3 * r2.a3);
126     produce M(Row(0, 0, r3.a3-k1*r1.a5-k2*r2.a4, r3.a4-k2*r2.a5, r3.a5, r3.rhs-k1*r1.rhs-k2*r2.rhs,
r3.seg));
127 }
128    group RIGHT_TO_LEFT:
129 // ----------------------------------------
130 M(r1) >> M(r2) M(r3) :
131 {   produce M(Row(0, 0, r1.a3, 0, 0, r1.rhs - r1.a4 * r2.rhs / r2.a3 - r1.a5 * r3.rhs / r3.a3, r1.seg));
132 }
133    group COLLECT:
```

```
// ---------------------------------------
B() < M(r) : // discard the first phony row
{   produce ;
}
M(r) > E() : // discard the last phony row
{   produce ;
}
M(r1) M(r2) : // convert two equations to a circuit segment
{   r1.seg.I = r1.rhs / r1.a3;
    r1.seg.V = r2.rhs / r2.a3;
    produce S (r1.seg);
}
E() :
{   curr_time += dt;
}

   interpretation:
   maximum depth: 1000;
// ============================================================
B() : // draw the intial voltage & resistor
{   nproduce SetWidth(2);
    x = -1;
    if (draw_circuit)
    {   nproduce SetColor(7)
            Line2d(V2d(-0.33,0),V2d(-0.33,1))
            Line2d(V2d(-0.33,1),V2d(0,1))
            Line2d(V2d(-0.33,0),V2d(0,0))
            ResistorV(-0.33,0.75,Rs)
            Emf (-0.33, 0.25, Vs);
    }
    produce SetColor(6) Line2d(V2d(0,1.2),V2d(0,1.2+vscale*Vs));
}
S(s) : // draw the segment and corresponding portion of the graph
{   x = x + 1;
    if (draw_circuit)
    {   static char buff1 [4096]; sprintf (buff1, "I=%.3f", s.I);
        static char buff2 [4096]; sprintf (buff2, "V=%.3f", s.V);
        nproduce SetColor (1)
            Line2d(V2d(x,1),V2d(x+1,1))
            Line2d(V2d(x+1,1),V2d(x+1,0))
            Line2d(V2d(x+0.5,0.5),V2d(x+1,0.5))
            Line2d(V2d(x+0.5,0.5),V2d(x+0.5,0))
            Line2d(V2d(x,0),V2d(x+1,0))
            Capacitor(x+1,0.25,s.Cap)
            ResistorV(x+0.5,0.25,s.Rp)
            ResistorV(x+1,0.75,s.Rv)
            ResistorH(x+0.5,1,s.Rh)
            SetColor(5) MoveTo(x+0.65,1.02,0) Label(buff1)
            SetColor(4) MoveTo(x+1.04,0.5,0) Label(buff2);
    }
    produceSetColor(6) Line2d(V2d(x,1.2),V2d(x+1,1.2))   // axis
        SetColor(5) RectangleF(x+0.5,1.2,x+1,1.2+vscale*s.I)// Render calculated current
        SetColor(4) RectangleF(x,1.2,x+0.5,1.2+vscale*s.V);// Render calculated voltage
}
E() : // draw the time
{   static char buff [4096]; sprintf (buff, "Time: %.3f", curr_time);
    produce SetColor(1) LabS (vscale,1.2,buff);
}
Capacitor(cx, cy, val) : // draw a capacitor
{   static char buff [4096]; sprintf (buff, "%.2f", val);
    produce SetColor(2) RectangleF(cx-0.1,cy-0.02,cx+0.1,cy+0.02)
        SetColor(1) Line2d(V2d(cx-0.1,cy-0.02),V2d(cx+0.1,cy-0.02))
        Line2d(V2d(cx-0.1,cy+0.02),V2d(cx+0.1,cy+0.02))
        SetColor(6) LabS(cx+0.02,cy+0.04,buff);
}
ResistorV(cx, cy, val) : // draw a vertical resistor
{   if (val == 0) produce ;
    static char buff [4096]; sprintf (buff, "%.2f", val);
```

```
202     produce SetColor(2) RectangleF(cx-0.02,cy-0.1,cx+0.02,cy+0.1)
203         SetColor(1) Rectangle (cx-0.02,cy-0.1,cx+0.02,cy+0.1)
204         SetColor(6) LabS(cx+0.04,cy,buff);
205 }
206 ResistorH(cx, cy, val) : // draw a horizontal resistor
207 {   if (val == 0) produce ;
208     static char buff [4096]; sprintf (buff, "%.2f", val);
209     produce SetColor(2) RectangleF(cx-0.1,cy-0.02,cx+0.1,cy+0.02)
210         SetColor(1) Rectangle (cx-0.1,cy-0.02,cx+0.1,cy+0.02)
211         SetColor(6) LabS(cx-0.1,cy+0.04,buff);
212 }
213 Emf cx, cy, val) : // draw EMF
214 {   static char buff [4096]; sprintf (buff, "%.2f", val);
215     produce SetColor(1) MoveTo(cx,cy,0) Circle(0.1) SetColor(2) Circle(0.09)
216         SetColor(6) LabS(cx+0.12,cy,buff);
217 }
218 RectangleF(x1, y1, x2, y2) : // draw filled rectangle
219 {   produce SP () MoveTo(x1,y1,0) PP() MoveTo(x2,y1,0) PP() MoveTo(x2,y2,0) PP()
220         MoveTo(x1,y2,0) PP() EP ();
221 }
222 Rectangle(x1, y1, x2, y2) : // draw outline of a rectangle
223 {   produce Line2d (V2d (x1, y1), V2d (x2, y1)) Line2d (V2d (x2, y1), V2d (x2, y2))
224         Line2d (V2d (x2, y2), V2d (x1, y2)) Line2d (V2d (x1, y2), V2d (x1, y1));
225 }
226 LabS(x, y, s) : // draw label
227 {   produce MoveTo(x,y,0) Label(s.c_str ());
228 }
```