The Hines Solver

This document describes the process of solving one timestep of the NEURON equations. First the equations and the algorithm for solving the linear system are presented using a mathematical formalism. Then snippets of code are analyzed to understand the actual implementation in NEURON code.

1 Mathematical representation

NEURON uses a discretized cable equation to compute voltage values along the neuron cell. A neuron is represented with a tree structure, where each element of the tree corresponds to a discretization node in the geometry (see Figure 1). An unbrached sequence of nodes is called a section (blue in Figure 1), and each node is called a compartment or segment (red in Figure 1). The discretized equations to be solved at node j are

$$\begin{aligned} c_j \frac{\Delta V_j^n - \Delta V_j^{n-1}}{\Delta t} - \frac{1}{2\pi a_j \Delta x}\left(\frac{1}{r_{j+1/2}}\frac{\Delta V_{j+1}^n - \Delta V_j^n}{\Delta x} + \frac{1}{r_{j-1/2}}\frac{\Delta V_{j-1}^n - \Delta V_j^n}{\Delta x}\right) + g_j \Delta V_j^n \\ = I_j^n + \frac{1}{2\pi a_j \Delta x}\left(\frac{1}{r_{j+1/2}}\frac{V_{j+1}^{n-1} - V_j^{n-1}}{\Delta x} + \frac{1}{r_{j-1/2}}\frac{V_{j-1}^{n-1} - V_j^{n-1}}{\Delta x}\right) - g_j V_j^{n-1}. \end{aligned} \tag{1}$$

Here the unknown is not the actual value, but rather the increment from the previous timestep. It is like that because the same equation could be used for

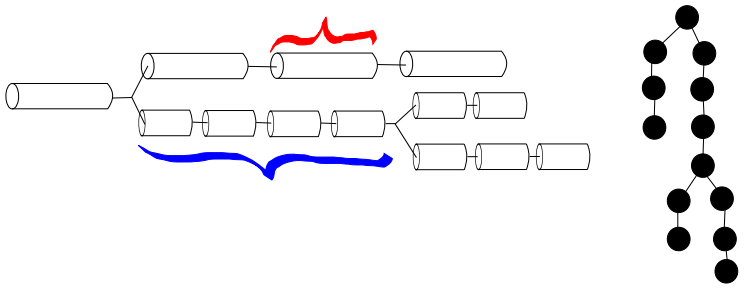

Figure 1: Cable representation (*left*) and tree representation (*right*). A section is highlighted in blue, and a single compartment (or segment) in red.

different time discretization schemes, e.g. Backward Euler and Crank Nicholson. In equation (1) we have $\Delta V_j^n = V_j^n - V_j^{n-1}$, $r_{j+1/2} = \frac{R_j}{2\pi(a_j^2+a_{j+1}^2)}$. Moreover, R (axial resistivity) is constant within each section, whereas a (radius) and c (membrane capacitance) can be different for each compartment. We leave some ambiguity in the definition of g_j in order to take into account passive currents and channels, while the term I_j^n is used to denote the stimulus. Considering a (node, parent, child) tuple (j, k, m) and injecting the respective definitions in the equation yields

$$\begin{aligned} c_j \frac{\Delta V_j^n - \Delta V_j^{n-1}}{\Delta t} - \frac{1}{2\pi a_j \Delta x}\left(\frac{2\pi(a_j^2+a_{j+1}^2)}{R}\frac{\Delta V_{j+1}^n - \Delta V_j^n}{\Delta x} + \frac{2\pi(a_j^2+a_{j-1}^2)}{R}\frac{\Delta V_{j-1}^n - \Delta V_j^n}{\Delta x}\right) + g_j \Delta V_j^n \\ = I_j^n + \frac{1}{2\pi a_j \Delta x}\left(\frac{2\pi(a_j^2+a_{j+1}^2)}{R}\frac{V_{j+1}^{n-1} - V_j^{n-1}}{\Delta x} + \frac{2\pi(a_j^2+a_{j-1}^2)}{R}\frac{V_{j-1}^{n-1} - V_j^{n-1}}{\Delta x}\right) - g_j V_j^{n-1}. \end{aligned} \tag{2}$$

1.1 Branching points

Branching points are a special case of equation (2). In particular, NEURON does not assign any channel/synapse to a branching point (i.e. $g_j,\ c_j = 0$). As a consequence of numbering (see Section 1.3), branching points are always at the end of a section. They inherit the diameter of the parent, the resistance value of the section and are assigned a value of 100 for the area. Therefore, equation (2) for a branching point reduces to (where m indexes all the different children)

$$\begin{aligned} -\frac{1}{100}\left(\frac{2\pi a_{j-1}^2}{R}\frac{\Delta V_{j-1}^n - \Delta V_j^n}{\Delta x}\right) - \sum_{m\in\text{children}} \frac{1}{100}\left(\frac{2\pi a_k^2}{R_k}\frac{\Delta V_k^n - \Delta V_j^n}{\Delta x}\right) \\ = \frac{1}{100}\left(\frac{2\pi a_{j-1}^2}{R}\frac{V_{j-1}^n - V_j^n}{\Delta x}\right) + \sum_{m\in\text{children}} \frac{1}{100}\left(\frac{2\pi a_k^2}{R_k}\frac{V_k^n - V_j^n}{\Delta x}\right). \end{aligned} \tag{3}$$

1.2 Units

A consideration should be made regarding units.

Property 1. *The following statements hold true for any NEURON simulation:*

- *all equations related to non-branching points are in* $\left[\frac{mA}{cm^2}\right]$*;*
- *all equations related non-branching points are in* $[nA]$*.*

Considering the NEURON GUI, we also have the following units for the various parameters:

V	$[mV]$	Δx	$[\mu m]$
c	$\left[\frac{\mu F}{cm^2}\right]$	Δt	$[s]$
a	$[\mu m]$	g	$[S]$
R	$[\Omega cm]$	I	$\left[\frac{mA}{cm^2}\right]$

Considering the coefficients appearing in (2), we then have:

$$\frac{c}{\Delta t} \qquad \left[\frac{\mu F}{s \cdot cm^2}\right] = \left[\frac{10^{-3} mA}{mV \cdot cm^2}\right]$$

$$\frac{1}{2\pi a_j \Delta x} \frac{2\pi(a_j^2 + a_{j+1}^2)}{R \Delta x} \qquad \left[\frac{1}{\mu m^2} \cdot \frac{\mu m^2}{\Omega cm \cdot \mu m}\right] = \left[\frac{mA}{mV \cdot 10^{-4} cm^2}\right]$$

The presence of these *hidden* powers of ten (10^{-3} for the capacitance and 10^{-4} for the resistance) makes us understand why dimensional analysis is important. Indeed, such coefficients are embedded in the NEURON code.

1.3 Numbering of nodes

The neuron is defined as a set of nodes connected by a *parent* relation. The Hines solver is special because the numbering of the nodes complies to a particular rule (see Figure 2):

Property 2. *the index of a parent cannot be larger than the index of its children.*

1.4 Matrix representation

Equation (1) can be split in four contributions: the interaction between nodes j and $j-1$, the interaction between nodes j and $j+1$, the self interacting terms for node j and the right hand side. The rearranged expression for equation (1)

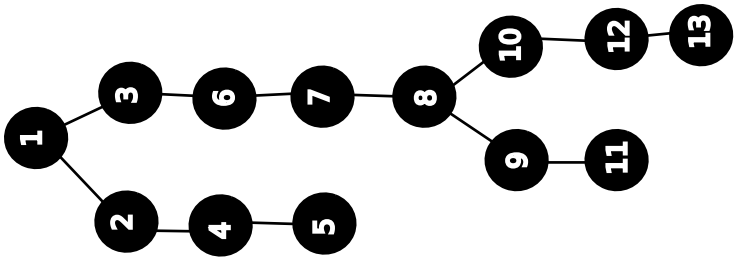

Figure 2: Example of numbering respecting property 2.

is:

$$\begin{aligned}
&\left(\frac{c_j}{\Delta t} + \frac{1}{2\pi a_j \Delta x^2}\left(\frac{1}{r_{j-1/2}} + \frac{1}{r_{j+1/2}}\right) + g_j\right)\Delta V_j^n + I_j^n \\
&\quad - \frac{1}{2\pi a_j \Delta x^2}\frac{1}{r_{j+1/2}}\Delta V_{j+1}^n - \frac{1}{2\pi a_j \Delta x^2}\frac{1}{r_{j-1/2}}\Delta V_{j-1}^n \\
&\quad = \frac{1}{2\pi a_j \Delta x}\left(\frac{1}{r_{j+1/2}}\frac{V_{j+1}^{n-1} - V_j^{n-1}}{\Delta x} + \frac{1}{r_{j-1/2}}\frac{V_{j-1}^{n-1} - V_j^{n-1}}{\Delta x}\right) - g_j V_j^{n-1} + \frac{c_j}{\Delta t}\Delta V_j^{n-1},
\end{aligned} \tag{4}$$

where the first line represents the self interacting terms inside node j (the definition of I_j may contain some terms that actually go in the right hand side, but should never contain cross-interaction terms with other nodes), the second line holds the $j, j+1$ and $j, j-1$ terms, and the last line the right hand side. Equation (4) can be generalized to take into account the tree representation of neurons if we consider $j-1$ as the parent of node j, and $j+1$ as the child of node j. Each of these equations is a current balance equation with units $\left[\frac{mA}{cm^2}\right]$. The cases where a node has multiple children (branching point) or no children at all (boundary node) are treated in a special way. For both, the current balance equations have units of nA, and they are assigned an area of 0. Moreover, to compute the $r_{j\pm1/2}$ values, only the parents or children's values are used. With this reasoning, the current balance equation for node j, with parent i and siblings indexed by k becomes (see also (5):

$$\begin{aligned}
&-\left(\frac{2\pi a_{j-1}^2}{R}\frac{\Delta V_{j-1}^n - \Delta V_j^n}{\Delta x}\right) - \sum_{m\in\text{children}}\left(\frac{2\pi a_k^2}{R_k}\frac{\Delta V_k^n - \Delta V_j^n}{\Delta x}\right) \\
&\qquad = \left(\frac{2\pi a_{j-1}^2}{R}\frac{V_{j-1}^n - V_j^n}{\Delta x}\right) + \sum_{m\in\text{children}}\left(\frac{2\pi a_k^2}{R_k}\frac{V_k^n - V_j^n}{\Delta x}\right).
\end{aligned} \tag{5}$$

Note that the fact of arbitrarily assigning a value 100 for the area of the branching point is not so important for the equation associated to the actual branching point, but rather for the node equations whose parent is a branching point.
If we view equation (4) as a set of coupled equations, with unknown the vector of voltage for different compartments at a given time step, then we can represent it with a matrix M. The following relation holds for a node-parent-child tuple (j, k, m):

$$\begin{aligned}
M_{j,j} &= \left(\frac{c_j}{\Delta t} + \frac{1}{2\pi a_j \Delta x^2}\left(\frac{1}{r_{j-1/2}} + \frac{1}{r_{j+1/2}}\right)\right) + g_j \\
M_{j,k} &= -\frac{1}{2\pi a_j \Delta x^2}\frac{1}{r_{j,k}} \\
M_{j,m} &= -\frac{1}{2\pi a_j \Delta x^2}\frac{1}{r_{j,m}}.
\end{aligned} \tag{6}$$

Injecting the definitions of $r_{l,n}$ and considering that we do not wish to store the information about the children, but only about the parents, relation (6) can be written in terms only of the node-parent couple (j,k) as:

$$\begin{aligned}
M_{j,j} &= \frac{c_j}{\Delta t} - M_{j,k} - \sum_{m \in \text{children(j)}} M_{j,m} + g_j \\
M_{j,k} &= -\frac{1}{2\pi a_j \Delta x^2} \left(\frac{2\pi(a_j^2 + a_k^2)}{R} \right) \\
M_{k,j} &= -\frac{1}{2\pi a_k \Delta x^2} \left(\frac{2\pi(a_j^2 + a_k^2)}{R} \right) \\
M_{j,m} &= -\frac{1}{2\pi a_j \Delta x^2} \left(\frac{2\pi(a_j^2 + a_m^2)}{R_m} \right)
\end{aligned} \tag{7}$$

Finally, taking into account the *hidden* powers of ten mentioned in Section 1.2, we have:

$$\begin{aligned}
M_{j,j} &= 10^{-3} \frac{c_j}{\Delta t} - M_{j,k} - \sum_{m \in \text{children(j)}} M_{j,m} + g_j \\
M_{j,k} &= -\frac{10^4}{2\pi a_j \Delta x^2} \left(\frac{2\pi(a_j^2 + a_k^2)}{R} \right) \\
M_{k,j} &= -\frac{10^4}{2\pi a_k \Delta x^2} \left(\frac{2\pi(a_j^2 + a_k^2)}{R} \right) \\
M_{j,m} &= -\frac{10^4}{2\pi a_j \Delta x^2} \left(\frac{2\pi(a_j^2 + a_m^2)}{R} \right)
\end{aligned} \tag{8}$$

Remember that if k or any of the children indexed by m are a branching point, the $area = 100$ rule will apply, and the associated radius a would be null.
If j is a branching point, then we simply have:

$$\begin{aligned}
M_{j,j} &= -M_{j,k} - \sum_{m \in \text{children(j)}} M_{j,m} \\
M_{j,k} &= -100? \left(\frac{2\pi a_k^2}{R\Delta x} \right) \\
M_{k,j} &= -\frac{10^4 a_k}{R\Delta x^2} \\
M_{j,m} &= -100? \left(\frac{2\pi a_k^2}{R_m \Delta x} \right)
\end{aligned} \tag{9}$$

1.5 Linear system resolution

Once the left hand side and right hand side of equation (4) have been computed, we can proceed to the resolution of the linear system. This is done through a

standard triangularization and backward substitution procedure equivalent to an LU decomposition. Moreover, we can exploit the parent-tree structure to perform this in a very efficient way. In particular, because of property 2, we know that for a node j with parent k, the term $M_{j,k}$ belongs to the lower triangular part of the matrix, and the term $M_{k,j}$ belongs to the upper triangular part of the matrix. The Algorithm 1 describes the procedure: node j with parent k

Algorithm 1 Triangularization and backward substitution

for $i = N, ..., 0$ **do**
  $M_{p(i),p(i)} = M_{p(i),p(i)} - M_{i,p(i)} \frac{M_{p(i),i}}{M_{i,i}}$
  $RHS_{p(i)} = RHS_{p(i)} - RHS_i \frac{M_{p(i),i}}{M_{i,i}}$
end for
for $i = 0, ..., N$ **do**
  $RHS_i = RHS_i - RHS_{p(i)} M_{i,p(i)}$
  $SOLUTION_i = \frac{RHS_i}{M_{p(i),p(i)}}$
end for

eliminates the upper triangular element $M_{k,j}$, modifying $M_{k,k}$ and RHS_k, then computes the j-th solution component by backward substitution.

1.6 From any matrix to the corresponding tree representation

Property 2 implies this for the matrix:

Property 3. *In the upper triangular of the matrix, there is only one nonzero entry for each column.*

This gives us a necessary criterion (and I think it is also sufficient) to determine if a given matrix is suitable to be represented in the NEURON parent-tree structure or not.
Moreover, we have that the row index of the only nonzero entry corresponds to the parent index of the node associated to the column. To be more clear, we fix a node j and look for the only nonzero entry in upper triangular of the matrix, for example let's say that it is $M_{k,j}$ (recall that because we are in the upper triangular matrix we have $k < j$). Then we can infer that node k is the parent of node j. This means that from the matrix we are able to directly infer the topology of the neuron cell.

2 Input & output

Once the mathematical model has been understood, it is time to look into the code and extract the lines where the algorithm is being carried out. The conceptual flow can be roughly divided in three phases, as described in Figure 3: as input the code takes a topological representation of the neuron plus all the

relevant biological data, it internally represents it with the parent-tree structure described above and computes the solution to the linear system.

First of all, it is important to understand the input data. As shown in Figure 1, each neuron cell is divided in sections, each one comprised of a variable number of segments. Moreover, each cell has one (and only one) root node, which should be the node with the lowest index in the whole cell. For each section, the values of R and c are constant. The value of a can be assigned to a section, and it is not yet clear how it is computed for each node.

2.1 Representation of the topology

The topology *of one cell* is represented as a list of sections. Each section contains a list of nodes and a list of children sections. There is only one section that is not the child of any other section, and that is called the root section. The numbering is performed following Algorithm 2: it makes use of a dynamic list of sections that gets updated at every iteration. At the beginning, the variable `sections` should point to the root section of the cell. At the end of the first iteration of the while loop, all of the root section's children are appended to the variable `sections`. At the end of the whole numbering process, `sections` will be a list of all the sections in the cell. A section should then be represented as an

Algorithm 2 Numbering of nodes for one cell

```
Require: sections
  inode = 1
  while sections != NULL do
    for node=sections→first_node,....,sections→last_node do
      node→index = inode
      inode++
    end for
    for child=sections→first_sibling,....,sections→last_sibling do
      sections→append(child)
      child→parent_node = inode
    end for
    sections = sections→next
  end while
```

object containing at least the following members:

- biological parameters: c, a, R, L;

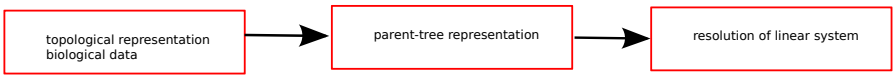

Figure 3: The three stages of the worfklow relevant to the linear algebra.

- number of segments;
- list of nodes, whose size is determined by the number of segments;
- list of children sections;
- index of parent node.

A node, on the other hand, should contain:

- index;
- index of parent;
- index of section it belongs to.

3 Input to vector representation

Now we aim at going from the list of sections (topological representation) of the previous algorithm to a data format composed of the following vectors:

- vector `v_parent_index` of parent indexes;
- vector `VEC_A` of upper triangular matrix values;
- vector `VEC_B` of lower triangular matrix values;
- vector `VEC_D` of diagonal matrix values;
- vector `VEC_RHS` of right hand side values.

To do so, we are going to use the previous list of sections (`sections`), and apply Algorithm 3 to fill the vector of parent indexes. The idea is that `v_parent_index[i]` contains the index of the parent of node i. Once the parent indexes have been filled, we may proceed to filling the vectors that represent the matrix using Algorithm 4. Note that in the filling of `VEC_D` we use the previously computed values for `VEC_A,VEC_B`. This is consistent with equation (6), once the correct indexing between parents has been figured out.

4 Vector representation to solution

Property 4. *the following relations hold:*

- *vector `VEC_A` holds the information regarding the upper triangular matrix. In particular, it holds the information regarding the effect of the node's voltage on the parent equation;*
- *vector `VEC_B` holds the information regarding the lower triangular matrix. In particular, it holds the information regarding the effect of the parent's voltage on the node equation;*

Algorithm 3 From topological to vector representation: parent indexes

```
Require: sections
  pnode = -1
  while sections != NULL do
    for node=sections→first_node,....,sections→last_node do
      if node==sections→first_node then
        v_parent_index[node→index] = sections→parent_node
        pnode = node→index
      else
        v_parent_index[node→index] = pnode
        pnode++
      end if
    end for
    sections = sections→next
  end while
```

- *vector* `VEC_D` *holds the information regarding the diagonal of the matrix. In particular, it holds the information regarding the interactions of a node with itself, the channels, the synapses, and any additional mechanisms like passive currents;*
- `VEC_A[i]` $= M_{parent(i),i}$*;*
- `VEC_B[i]` $= M_{i,parent(i)}$*;*
- `VEC_D[i]` $= M_{i,i}$*;*

Keeping Property 4 in mind, the triangularization procedure can be written as in Algorithm 5 Note that at the end triangularization process, we expect the vector `VEC_A` to only contain zeros. However, we do not need to explicitly enforce this, because thanks to the particular matrix representation we are using, we can then simply top using `VEC_A` in the rest of the process, effectively avoiding the need to concretely put the values to zero. The Backward substitution is described in Algorithm 6. One important side effect of this algorithm is that the final result is stored in `VEC_RHS`, thus destroying the the right hand side values. A second important side effect to consider is that `VEC_A` and `VEC_B` are not modified by the algorithm. This is also synergetic with equation (6): the out-of-diagonal entries of matrix are also not time-dependent, so that these vectors are effectively constant throughout the whole simulation.

5 References to NEURON code

Algorithms 2 and 3 for numbering the nodes and the parent indexes respectively can be found in the file `nrn/src/nrnoc/multicore.c`, in function `reorder_secorder()` around line 760.
Algorithm 4 for populating the vectors can be found in the file `nrn/src/nrnoc/treeset.c`,

Algorithm 4 Filling the vectors that represent the matrix

Require: v_parent_index
  while sections != NULL **do**
    $\Delta x = \frac{L}{NumberOfSegments}$
    $R_{right} = 0$
    for node=sections→first_node,....,sections→secondlast_node **do**
      AREA[node→index] = $2a\pi\Delta x$
      $R_{left} = R\frac{\Delta x}{2\pi a^2}$
      NODERINV[node→index] = $\frac{1}{R_{left}+R_{right}}$
      $R_{right} = R_{left}$
    end for
    AREA[sections→last_node→index] = 100
    NODERINV[sections→last_node→index] = $\frac{1}{R_{right}}$
    sections = sections→next
  end while
  while sections != NULL **do**
    area = AREA[sections→parent_node]
    for node=sections→first_node,....,sections→secondlast_node **do**
      VEC_A[node→index] = $\frac{\text{NODERINV[node→index]}}{\text{area}}$
      area = AREA[node→index]
      VEC_B[node→index] = $\frac{\text{NODERINV[node→index]}}{\text{area}}$
    end for
    VEC_D = 0
    for node=sections→first_node,....,sections→secondlast_node **do**
      VEC_D[node→index] -= VEC_B[node→index]
      VEC_D[v_parent_index[node→index]] -= VEC_A[node→index]
    end for
    sections = sections→next
  end while

Algorithm 5 Triangularization

Require: TotalNumberOfNodes, VEC_A, VEC_B, VEC_D, VEC_RHS, v_parent_index
  for i=TotalNumberOfNodes,....,0 **do**
    VEC_D(v_parent_index[i]) -= $\frac{VEC\_A[i]}{VEC\_D[i]}VEC\_B[i]$
    VEC_RHS(v_parent_index[i]) -= $\frac{VEC\_A[i]}{VEC\_D[i]}VEC\_RHS[i]$
  end for

Algorithm 6 Backward substitution

Require: TotalNumberOfNodes, VEC_B, VEC_D, VEC_RHS, v_parent_index
  for i=0,...,TotalNumberOfNodes **do**
    VEC_RHS[i] -= VEC_B[i] * VEC_RHS[v_parent_index[i]];
    VEC_RHS[i] /= VEC_D[i];
  end for

in function `nrn_area_ri()` around line 710.
The triangularization and backward substitution algorithms can be found in the file `nrn/src/nrnoc/solve.c`, in functions `triang(NrnThread* _nt), bksub(NrnThread* _nt)` around line 380.