**Final Report:**

**A Direct Linear Sparse-Matrix Solver Implementation in Screamer**

Dr. Rick B. Spielman and Dr. Yury Gryazin Idaho State University

Pocatello, ID

November 4, 2014

# Introduction

We have replaced the legacy, sparse-matrix solver subroutine in Screamer, Sandia National Laboratories’ versatile circuit code. We moved from a optimized pentadiagonal/very sparse matrix solver (very fast but very limiting) to a more general, sparse-matrix solver that is based on a linear, Gaussian matrix-solver algorithm (2X slower but with more general capabilities). Our goal was to take advantage of the faster CPUs and larger, dynamic random- access (DRAM) memories found in modern personal computers and to use Screamer for more complex circuit problems. The new subroutines are very clearly written and well documented. The structure of the problem matrix and the solution vector is now explicit rather than obscure and implicit.

Screamer was originally written in Fortran 77 (f77) for use on the DEC VAX platforms. Our decision to continue to use f77 allowed us to tap an extensive library of matrix solver subroutines and to take advantage of the extensive experience base in conducting matrix mathematics. Finally, the many existing Fortran 77 matrix solvers are all highly optimized for speed and often form the core of modern super-computer algorithms.

We have conducted benchmarks of Screamer V4.0 against Screamer V3.3.2 using a wide variety of test circuit problems. We have found that Screamer V4.0 averages 2X slower than Screamer V3.3.2 while requiring up to ~10X more memory. We deliberately traded off memory size to a certain extent in order to populate the problem arrays and vectors and to build a very clear matrix representation of the problem and solution.

# Screamer Background and History

In the mid 1980’s, it became apparent that a new circuit code was needed to address a set of specific and critical needs that the Pulsed Power Center at Sandia National Laboratories had in modeling complex electrical circuits. The design of new, pulsed power systems was being slowed down by our inability to rapidly model existing drivers at Sandia (and in other laboratories) and to design new drivers. In the early and mid 1980’s the circuit code that was used at Sandia was Sceptre.1 It ran on Sandia’s Cray-1, then one of the fastest computers in the country. For example, a complex, high-accuracy run modeling the Sandia Comet accelerator took 42 minutes of Cray CPU time and had a 2-day turn around time.2 At that time, the Cray was heavily subscribed. In addition, it was a difficult and time-consuming task to run the Sceptre code. This made it difficult for experimental scientists to model their own pulsed-power machines. This need for a more usable and dedicated circuit code drove a project at Sandia to build a code that was smaller, much faster, and more user friendly. These new code’s operational needs lead to the following design goals.

First, any code that was developed had to be much faster than the existing Sceptre code and use much less computer memory. This meant that serious trade-offs had to be made, in which limited circuit architecture flexibility was balanced against speed and memory. The computer platform that was chosen to run Screamer was the DEC VAX 780 series of computers that were then being installed at Sandia. It was imperative that Sandia reduce the run time of circuit problems from days to hours.

Second, the code had to include a growing number of analytic and heuristic models that were needed to describe the performance of a wide range of pulsed-power drivers that were needed. This included various types of switching, vacuum transmission lines, and dynamic loads (ion beam, electron beam, and z pinches).

Third, the resulting code had to be able to be run by pulsed-power scientists and engineers – not computational and theoretical physicists. This tool was to be widely used by the very scientists designing the new pulsed-power drivers.

Finally, the code had to be easily modifiable. Scientists and engineers had to be able to work

with the code designers to implement new capabilities rapidly.

The result of these carefully considered constraints was Screamer2. It was a fast, accurate user-friendly, and flexible. The sparse, “tri-diagonal” matrix solver used scaled linearly as the number of problem nodes. Screamer was 100X faster than similar runs made on the Cray and became a standard for machine design at Sandia National Laboratories. We will discuss the legacy Screamer sparse-matrix solver in more detail later in this report.

## Legacy Screamer Limitations

Screamer deliberately limited the architectures and circuit sizes that could be modeled to dramatically increase the speed of the mathematics. The following design choices all contributed to the speed and compactness of Screamer.

* + 1. Screamer chose a fundamental circuit structure that used series resistors and inductors and parallel resistors (conductance) and capacitances to ground (a series of pi-blocks). This simple structure could easily include transmission lines, variable resistors, switches, etc. All of these circuit elements could be simply modeled in terms of voltages and currents.
    2. Screamer limited branch structures severely. With some very tight restrictions, controlled branching was allowed. The number of branches was limited to control speed and memory.
    3. The total number of nodes was limited to a rather large number to control memory requirements.
    4. Branches in branches were forbidden. This allowed the basic solution matrix to be a sparse tri-diagonal matrix. This was critically necessary to control the run time of the code by allowing a sparse matrix solver.
    5. A fully optimized, custom sparse-matrix solver was written.2
    6. Screamer was written in Fortran 77 for absolute speed.

# Screamer Circuit Solver Analysis

From first principles, we can derive the fundamental equations that were (and are) used in Screamer. First, we draw the layout of two arbitrary Screamer nodes for reference.

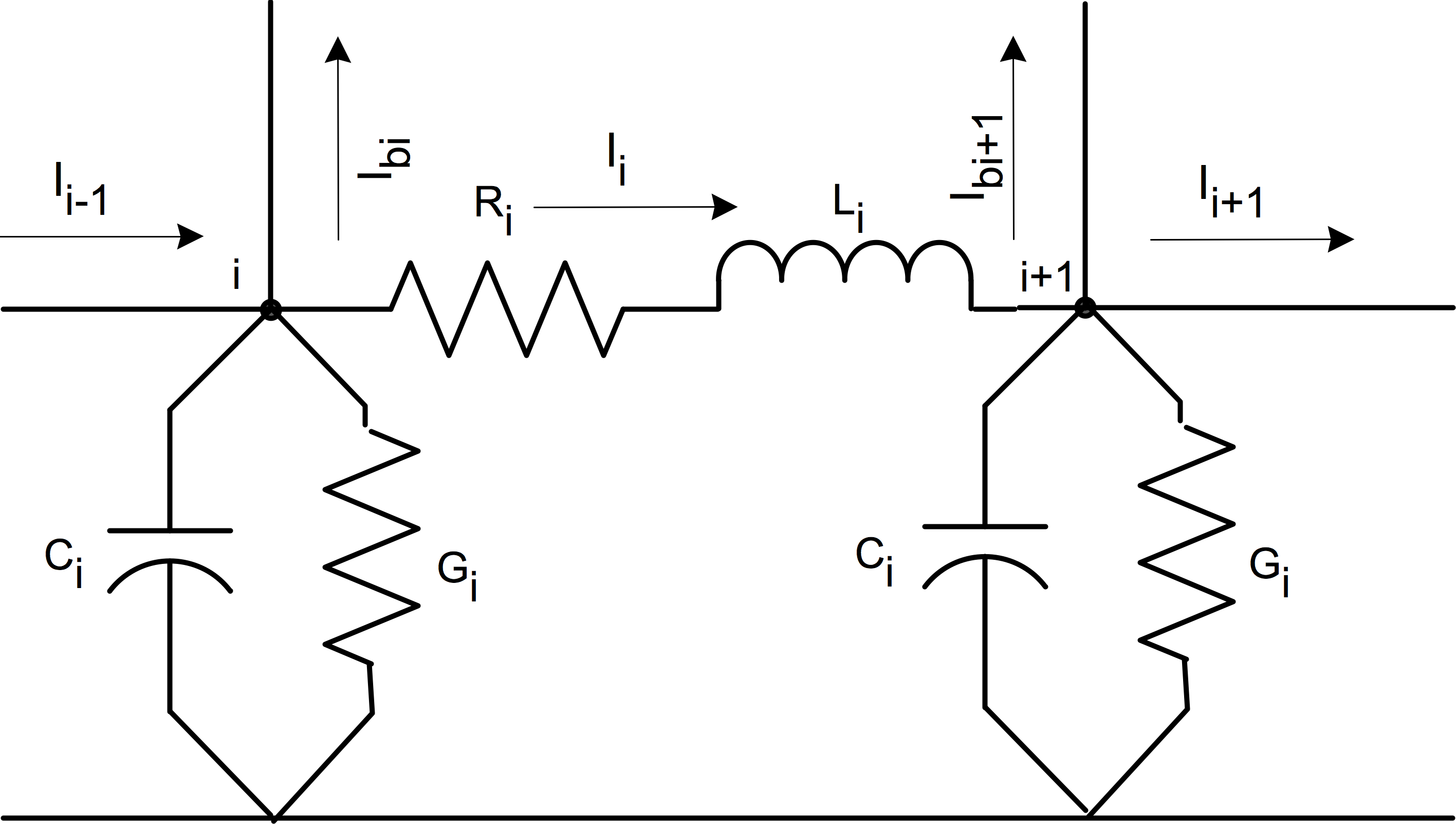


Fig. 3.1 A two-node example of Screamer elements

We can write the basic current equation of a single node. The current into the *ith* node (from the *ith* – 1 node) equals the current out of the *ith* node in the main branch, plus the current out of the *ith* node into the branch *b* at the *ith* node, and plus the current that flows through the circuit elements to ground from the *ith* node. Screamer uses the conductance *G* (1/*R*) as the resistive path to ground and the capacitance *C* as the capacitance to ground. These currents are shown in Eq. 3.1 below.

*Ii*1 = *Ii* + *Ibi* + *GiVi* + 

*t*

*CiVi*

(3.1)

We can then write the voltage equation between the *ith* and *ith* +1 nodes. The voltage drop between the *ith* node and the *ith* + 1 node is only the voltage drop across the *ith* resistor and *ith* inductor due to the *ith* current. These voltages are shown in Eq. 3.2 below.

*Vi*  *Vi*+1 = *RiIi* + 

*t*

*LiIi*

(3.2)

We can then express both equations in terms of half time steps in which the values of the current and voltage from the old time step (*o*) are known and the values at the new time step

(*n*) are to be calculated. The use of split time steps increases the fundamental accuracy of the calculation. The voltage and current expressions for half time steps are below in Eq. 3.3.

*n o n o n o*

*Vi* = 0.5 *Vi* + *Vi* , *Ii* = 0.5 *Ii* + *Ii* , *and Ibi* = 0.5 *Ibi* + *Ibi*

(3.3)

The partial derivative of (*LI*) and (*CV*) with time is simply expressed as a difference in the values of *I* and of *V* divided by the time step. This is shown in Eqs. 3.4 & 3.5. Note the use of the difference in current and voltage in the partial derivative with the assumption that *L* and *C* are not functions of time.

*n o*

 *CV* = *C*  *V* = *C V*  *V*

*t* *t t* (3.4)

*n o*

 *LI* = *L*  *I* = *C I*  *I*

*t* *t t* (3.5)

We can now express Eqs. 3.1 & 3.2 in terms of new and old variables. It is instructive to note that the presence of a branch at the *ith* node only shows up in the *ith* current equation of the main branch. (This will directly lead to a sparse matrix when branches are used.) We collect all of the new variables on the left and the old variables on the right. The current equation is shown in Eq. 3.6 and the voltage equation is shown in Eq. 3.7.

*Ci*

*t*

*n o*

0.5 *I*  *I*

*n o n o*

*n o n o*

*i*1

*i*1

 0.5 *Ii*

* *Ii*
  + 0.5 *Ibi* + *Ibi*

= 0.5*Gi Vi*

* *Vi* +

*Vi*  *Vi*

*n n n*

*Ci*

*t*

0.5 *I*

*n o o o o*

*i*1  *Ii*

*Ci*

*t*

* *Ibi*

 0.5*Gi* +

*Vi* = 0.5 *Ii*1  *Ii*

* *Ibi*

+ 0.5*Gi*  *Vi*

(3.6)

*n o n o n o n o*

*Li*

*�t*

0.5 *Vi*  *Vi*  0.5 *Vi*+1  *Vi*+1 = 0.5*Ri Ii*  *Ii* + *Ii* *Ii*

*n n n o o o*

*Li*

*�t*

*Li*

*�t*

0.5 *Vi*

 *Vi*+1

 0.5*Ri* +

*Ii* = 0.5 *Vi*1  *Vi*

+ 0.5*Ri*  *Ii*

(3.7)

The RHS’s of both equations are constants that are derived only from the circuit variables in prior time steps. The circuit values on the LHS of the equations in the parentheses are also constants. First, let’s rewrite Eqs. 3.6 & 3.7.

*Ci*

*t*

*Ci*

*t*

*n n n*

0.5 -*I*

*n o o o o*

*i*  1  *Ii*

* *Ibi*

 0.5*Gi* 

*Vi* = 0.5 *Ii*1  *Ii*

* *Ibi* 
  + 0.5*Gi Vi*

(3.8)

*n*

0.5 *Vi*

*n*

 *Vi*+1

 0.5*Ri* 

*n o*

*Ii* = 0.5 *Vi*

*Li*

*t*

*o*

 *Vi*+1 

*o*

* 0.5*Ri Ii*

*Li*

*t*

(3.9)

We can further simplify the form of the equations by defining the four constants separately in Eq. 3.10. We use exactly these four constants in Screamer. The *Ai* constants depend only on circuit values. The *Bi* constants can be populated with circuit values and from the results of the prior time step.

*AVi* = 0.5*Gi* 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Ci* | *o o o*  *and BV* = 0.5 *I*  *I*  *I*  | | *Ci* | *o*   0.5*G V* |
| *t*  *Li* | *i i*1 *i bi*  *o o Li*  *and BI* = 0.5 *V*  *V*  |  | *t*  0. | *i i*  *o*  5*R I* |
| *t* | *i i i*+1 *t* |  |  | *i i* |

*AIi*

= 0.5*Ri* 

(3.10)

By substituting these variables into Eqs. 3.8 & 3.9 we see that we have a simple set of 2•*N* equations and 2•*N* unknowns. Where *N* is the total number of nodes and *i* = 1, *N*. Thus, we finally have the simple, linear-equation representation of the circuit equations in Eqs. 3.11 &

3.12. Each of these equations is recalculated moving forward time step by time step and we leave out the notation indicating a specific time step for clarity.

*n n n n*

0.5 *Ii*1  *Ii*  *Ibi*  *AViVi*

= *BVi*

(3.11)

*n*

0.5 *Vi*

*n*

* *Vi*+1

*n*

* *AIiIi*

= *BIi*

(3.12)

We can take the coupled set of Eqs. 3.11 & 3.12 and rewrite them in terms of alternating current and voltage with increasing node index *i* in Eqs. 3.13 & 3.14. Finally, for simplification of the notation, we will drop the superscript *n* (which was indicative of the “new” value of voltage, which is now assumed to be calculated), add a superscript *k* indicative of the branch number, where *k* = 1, *nb* and *nb* is the total number of branches.

*k k k k k k*

*Ii*1  *AVi Vi*  0.5*Ii*  0.5*Ib l*,*i* = *BVi*

(3.13)

*k k k k k*

*Vi*  *AIi Ii*  0.5*Vi*+1 = *BIi*

(3.14)

For only the case of i > 1 and I < *nk* (*i*), we multiply both Eqs. 3.13 & 3.14 by 2 and Eq. 3.14 equation by -1. While these choices seem arbitrary at this point, we have determined that this reduces number of calculations in the solver. The constant *nb* remains the total number of branches in the problem. Eq. 3.13 is the current equation for the *kth* branch and in the *ith* node of that *kth* branch, current leaves the *kth* branch into the first node a new *lth* branch. In Screamer, the branch index inequality *k* < *l* must hold. (In the limitation for the legacy solver, branches cannot have branches – except branch *k* = 1 – and all of the current and voltage equations for the branches (*k* > 1) do NOT have a branch term.)

*k k k k l k*

*I*

(3.15)

*i*1  2*AVi Vi*  *Ii*  *I*1 = 2*BVi*

*k k k k k*

(3.16)

*Vi*  2*AIi Ii*  *Vi*+1 = 2*BIi*

Eqs. 3.15 & 3.16 are the equations we will use to populate the full problem matrix and vector in the section below for *i* > 1 and *i* < *nk*(*i*).

## Example Matrix Population

Assume a simple example with no branches. We can express this simple problem with three

nodes as a vector with 6 elements in the matrix formation shown in Eqs. 3.13 – 3.16. The *AVi* and *AIi* constants are used to populate the ***A*** matrix and the *BVi* and *BIi* constants are used to populate the ***b*** vector. The first row has no current from the prior node and the last row has no voltage in the following node. This is true even with branches. Be careful with the first and last nodes of all branches, they are based on Eqs. 3.13 & 3.14.

1

*AV*1 .5 0 0 0 0

1

.5 *AI*1 .5 0 0 0

1

0 1 2*AV* 2 1 0 0

1

1 1

*V*1 *BV*1

1 1

*I* 1 *BI* 1

1 1

*V*2 2*BV* 2

=

1 1

0 0 1 2*AI* 2 1 0

1

*I* 2 2*BI* 2

1 1

0 0 0 .5 *AV*3 .5 *V*3

1 1

*BV*3

1

0 0 0 0 .5 *AI*3 *I* 3

*BI* 3

(3.17)

We can express this matrix and the two vectors in simple matrix notation. Where the ***x*** vector is the solution vector.

***A*** • ***x*** = ***b*** (3.18)

## End Branch Analysis

The end branch assumptions are that the input voltage to the *lth* branch equals the voltage at the exiting *ith* node in the main branch and that the branch current leaving the main branch must equal the branch current at the first node of the end branch.

*k l*

*VBl* = *V*1

(3.19)

*k l*

*IBl* = *I*1

(3.20)

where this is the first element in the (*lth*) end branch in the ***A*** matrix diagonal where its location in the main ***A*** matrix is at (2 • *nr*(*l -* 1) + 1, 2 • *nr*(*l* - 1) + 1), where *nr*(*k*), *k* = 1 is the number of nodes in the main (first) branch. The elements of the end branch are the same as a normal branch except for the first row because there are no branches off of branches. The other coupling coefficients show up in the row of the main branch in which the end branch leaves and in the row of the start of the end branch.

Let’s start with a very simple example with a main branch with three nodes (*N* = 3) or a vector with 6 (2 • *N*) elements and an end branch with two nodes and three elements. The base matrix without a branch is shown in Eq. 3.17. The fundamental current equation of a main branch node having a branch is Eq. 3.15 for *i* > 1 and *i* < *nk*(*i*):

1. *k k k l k*

*I*

(3.15)

*i*1  2*AVi Vi*  *Ii*  *I*1 = 2*BVi*

We can now see clearly how a branch places an off diagonal element in the ***A*** matrix. When this matrix row is multiplied by the solution vector ***x*** the off diagonal +1 array element multiplies the first current element in the branch. A little work shows that the first node in the *lth* branch cannot have an RCG element because that would result in a *BV11* constant that is not simply amenable to the planned matrix implementation.

Given the simplicity here, we can use Eqs. 3.13, 3.14, 3.15, and 3.16 to populate our example matrix. If we rewrite the current equation for a single end branch at node 2 of the main branch we can see how all of the matrix elements are created. The first four terms of the equation are the same as a main branch w/o branches but the last term is the off diagonal term that shows up in the ***A*** matrix due to the end branch. The off diagonal term for *I21* is the branch current coupling coefficient. This means that row 3 will have the following elements:

0 -1 2*AV2* 1 0 0 0 1 0 0

Note that the voltage in the main branch node 2 must equal the voltage in branch 1 node 1 from Eq. 3.19. This places -1 in row 7, column 3. This also forces a 0 in the ***b*** vector in the 7th location. (Row 7 of the ***A*** matrix times the ***x*** vector must equal element 7 of the ***b*** vector.)

We now get the entire ***A*** matrix with a single end branch.

1

*AV*1

.5 0 0 0 0 0 0 0 0 *V*1

1 1

1

1

*BV*1

1

 *AI*1 .5 0 0 0 0 0 0 0 *I*1

1 1

*BI*1

1

0  2*AV* 2 1 0 0 0 1 0 0 *V*2

1 1

2*BV* 2

1

0 0 1 2*AI* 2 1 0 0 0 0 0

1

*I*2 2*BI* 2

1 1

0 0 0 .5 *AV*3

.5 0 0 0 0

1

*V*3 *BV*3

1 1

=

0 0 0 0 .5 *AI*3

0 0 0 0 *I*3

*BI*3

0 0 1 0 0 0 1 0 0 0 *V* 0

2

1

1

2 2

0 0 0 0 0 0 .5 *AI*1 .5 0 *I*1

2 2

0 0 0 0 0 0 0 .5 *AV*2 .5 *V*2

2 2

0 0 0 0 0 0 0 0 .5 *AI*2 *I*2

*BI*1

1

*BV*2

1

*BI*2

(3.21)

First, note that the last current equation of the main branch does not have the 1 following the diagonal element. This is because it is the last node in the main branch and there is no voltage in the next node to the right because there is no next node to the right in that branch.

Thus, we can see that the addition of a single end branch to the main branch at the *ith* node impacts the row from which the end branch leaves (2•*ith* node-1) and the row of the first node (voltage) of the end branch. The first node of the end branch starts at the row and column 2•*nr*(1)+1, where *nr*(1) is the number of nodes in the main branch. Please note again that the matrix structure of the end branch is *exactly* the same as that of the main branch with the exception of the first node in the end branch. This means that the algorithm that populates the matrix with the end branch information can be the same algorithm as the main branch algorithm (except for the first node).

## Top Branch Analysis

A top branch can only exit the main branch *across* a series element. In this case, the voltage across the first branch node is the voltage across the series element. Current leaves from the *ith* node and returns in the *ith*+1 node. This means that the current equations for the *ith* and *ith*+1 nodes in the *kth* branch must include the branch currents leaving and entering the main branch (entering and leaving the *lth* branch). And addition, we have the following relationships as well.

1. *k k*

*V*1 = *Vi*

*k*

* *Vi* +1

*k*

*IBli* = *IBli* +1

(3.22)

Where the branch identifier is the superscript *k* and the subscripts are the main branch nodes from which the branch exits and enters. The individual branch currents are defined in a top- branch analysis in the same way as they were in Eq. 3.18. Because of the current relationship in Eq. 3.22, the branch current in the following node is the negative of the branch current in the prior node and both equations point to the same off diagonal column number. The voltage relationship defines the first node of the top branch.

Let’s consider an example having four nodes with the top branch departing from node 2 and returning on node 3. (Node 4 will be an RCground to end the main branch.) We added node 4 just to completely clarify the separation of the matrix elements.

First, the current equation at node 2 is below. Second, we have the current equation at node 3 where we replace the node-3 branch current with the negative of the node-2 branch current. We now have the coupling coefficients for the current equations. These are straight forward and are exactly the same as the end branch. We simply have 2 current equations to handle with off-diagonal elements.

In Row 3 we have exactly the same coefficient as the side branch!

0 -1 2*AV2* 1 0 0 0 0 0 1 0 0

In Row 5 we have nearly the same as row three except the coefficient is negative and we use the appropriate *AV3*.

0 -1 2*AV3* 1 0 0 0 0 0 -1 0 0

We need to include the coupling coefficients for the voltage terms. The voltage at the first node of the branch must be the difference of the voltage in the main branch at nodes 2 and 3, Eq. 3.19. To get this result we need a 1 at *V2* and a -1 at *V3* and a zero at *BV11* (just like the end branch). We now get the entire ***A*** matrix with a single top branch.

1

*AV*1

.5 0 0 0 0 0 0 0 0 0 0 *V*1

1 1

1

1

*BV*1

1

.5 *AI*1 .5 0 0 0 0 0 0 0 0 0 *I*1

1 1

*BI*1

1

0 1 2*AV* 2 1 0 0 0 0 0 1 0 0 *V*2

1 1

2*BV* 2

1

0 0 1 2*AI* 2

1 0 0 0 0 0 0 0

1

*I*2 2*BI* 2

1 1

0 0 0 1 2*AV* 3 1 0 0 0 1 0 0 *V*3

1 1

2*BV* 3

1

0 0 0 0 1 2*AI* 3

1 0 0 0 0 0

1

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | .5 | *AV*4 | .5 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | .5 | 1  *AI*4 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

2

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | .5 | *AV* | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | .5 | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

.5 0

2

*AI*2 .5

2

.5 *AV*2

*I*3 2*BI* 3

1 1

=

*V*4 *BV*4

1 1

*I*4 *BI*4

2 0

*V*

1 2

2 *BI*2

*I*

1 2

2 *BV*2

*V*

2 2

2 *BI*2

*I*

2

(3.23)

Again, note that the last current equation of the main branch (*k* = 1) does not have the 1 following the diagonal element. This is because it is the last node in the main branch and there is no voltage in the next node to the right because there is no next node to the right.

We can see that the addition of a single top branch to the main branch at the *ith* node impacts the row from which the top branch leaves (2•*ith* node - 1), the node to which the top branch enters (2•*ith* node) and the row of the first node (voltage equality) of the top branch. The first node of the top branch starts at the row and column 2•*nr*(1) + 1, where *nr*(1) is the number of nodes in the main branch. Please note again that the matrix structure of the top branch is the same as that of the main branch with the exception of the first node in the top branch. This means that the algorithm that populates the matrix with the top branch information can be the same algorithm as the main branch algorithm (except for the first node).

# How Legacy Screamer Builds The Actual *A* Matrix and *b* vector

The discussion above is mathematically correct but Screamer actually generates a slightly different node structure and different node counts than the simple discussion of circuit elements above might suggest. Building a circuit from Screamer circuit blocks requires open nodes for connectivity to the next block. In practice, this means that we need *every* circuit block to end in a “dangling” resistor/inductor series block (RLseries). With this circuit configuration one can connect any type of circuit block in series. (Consider the trivial case if you tried to connect two RCgrounds one after the other.)

Screamer, as written, creates an extra *phantom* block and nodes that separate actual circuit elements. For example, a simple resistor & capacitor to ground block (RCground) mathematically requires only a single *ith* node and the *G* & *C* elements to ground but Screamer adds a phantom RLseries element following the *ith* node. In this case, the *R* & *L* in the *ith* phantom RLseries element are set to zero.

Similarly, a resistor and inductor in series block (RLseries) would naturally be placed between the *ith* and *ith*+1 nodes. Phantom RCground elements are placed at the *ith* and *ith*+1 nodes. A phantom RLseries element follows the *ith* + 1 node just as is done for the RCground block. In this case, the *G*’s (1/*R*) & *C*’s in the RCgrounds are set to zero and the *R* & *L* in the second, phantom, RLseries are set to zero. Only the first RLseries element has non-zero values.

For a simple example consisting of circuit elements RCground/RLseries/RLseries/RCground there are four nodes required in the simple derivation in Section 3. In the actual Screamer code, however, there are six nodes created.

Node 1 – RCground (In the dangling phantom RLseries, *R* & *L* are 0.)

Nodes 2 & 3 RLseries (A phantom RCground is off of each node and a dangling phantom RLseries follows the second node, all *C*’s & *G*’s are 0 and the *R* & *L* in the phantom RLseries following the second node are zero.)

Nodes 4 & 5 RLseries (A phantom RCground is off of each node and a dangling phantom

RLseries follows the second node, all *C*’s & *G*’s are 0 and the *R* & *L* in the phantom RLseries following the second node are zero.)

Node 6 – A final RCground block (In the dangling phantom RLS, *R* & *L* are 0.)

Fig. 1 below shows the nodes and elements of this simple example problem (without branches) that would be solved in Screamer.

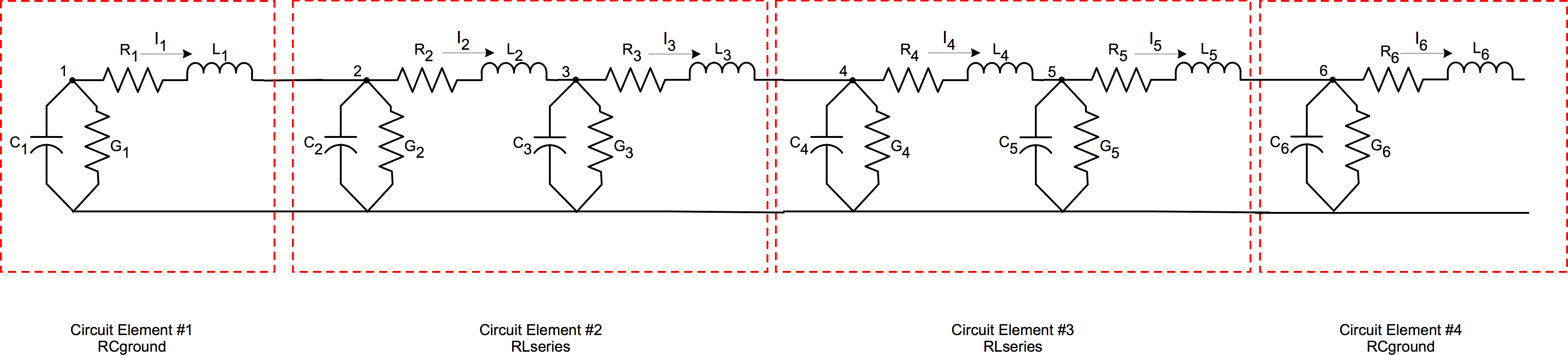


Figure 1 Node structure for a simple test file showing the phantom blocks and the dangling RLseries block.

There are currents, voltages, *A* coefficients, and *B* coefficients for each *ith* node. However, many of the *AVi* and *AIi* coefficients are 0. (This will lead to mathematics problems later for the matrix solution.) We describe the node structure of this sample problem in more detail below.

With an RCground at node 1, there is no *actual* series element between nodes 1 and 2 to connect to the next block, so after node 1 a *phantom* RLS must be included, in which the RLseries parameters *L1* and *R1* are both set to 0. This is the way that Screamer puts a connecting “wire” in the circuit. From the voltage equation, this means that *V(i)* = *V(i+1)* for all time steps or *V(i+1) - V(i)* = *0* for all time steps. Always think of an RCground block as if there is a phantom RLseries element with the *Ri* & *Li* values all zero in node *i*. In the current equation for *i*, the values of *Gi* and *Ci* are handled as usual from the inputted values.

Screamer places the next element, an RLseries block, between nodes 2 & 3. We see what are effectively two phantom RCground elements with their contents, *C* & *G*, set to zero and a dangling phantom RLseries block with all of its parameters zero following node 3. If you think about this for a bit you will understand that the requirement for a dangling phantom RLseries block leads naturally to the phantom RCground blocks. If one did not place an

official node immediately after the RLseries block after node 2 then the dangling RLseries phantom block would have to become part of the input RLseries block. It becomes simpler from a coding standpoint to use these phantom blocks with real nodes. The first RCground is at the 2*nd* node and the second RCground is at the 3*rd* node. Thus, a Screamer RLseries block actually populates ALL of the nodes that are available in that block and ends the block with a dangling phantom RLseries element.

Screamer places the next element, an RLseries block, between nodes 4 & 5. We see what are effectively two phantom RCground elements with their contents, *C* & *G*, set to zero and a dangling phantom RLseries block with all of its parameters zero following node 5.

Screamer places the final element, an RCground block, at node 6. Notice that node 6 is followed by a single phantom RLseries element that is always populated to zero values.

In the end, all this does is greatly simplify the way that arbitrary circuit elements can be tied together without consideration for internal code quirks. This has the very unfortunate effect of increasing the number of nodes that a full matrix solver will have to handle AND, more unfortunately, forcing many of the *A* and *B* constants to have a value of zero.

## Transmission Line Blocks in Screamer

In addition to the RLseries block and the RCground block, the transmission line block (TRline) is the key circuit element missing from our discussions. By now it should be obvious that a transmission line block is simply an RLseries block (pi-block) in which the *C*’s of the phantom RCground blocks are non-zero. Like an RLseries block, a TRline block is followed by a phantom RLseries block. The *L* of the main circuit element and the *C*’s of the RCground elements are determined by the specified transmission line impedance *Z* (Ω) and transmission-line length *t* (s). All of the phantom block circuit values are set to 0.

To get the appropriate time resolution for the TRline element we can break the single RLseries block into many blocks with the appropriate *L*’s and *C*’s to maintain the local transmission line impedance Z while having a shorter length *t*. Please carefully note that this increases the number of nodes in the problem. For example, a 100-ns long TRline block only needs 2 nodes if the resolution is set to 100 ns. If we set the resolution to 1 ns then we break

the TRline into 100 smaller TRlines and go from 2 nodes to 200 nodes.

C = *t* / Z (4.1)

L = *t* • Z (4.2)

Note: one of the great strengths of Screamer is its ability to break down the length of a transmission line into elements having durations smaller than the fundamental time step of the code. This allows us to fully resolve the transmission line.

## Populating the *A* matrix & *b* vector

The *A* constants are based on the (usually fixed) values of the circuit parameters. These values are used to populate the elements of the ***A*** matrix. The ***b*** vector is the vector that contains the circuit information from the prior time step as well as the values of circuit parameters. This includes fixed or variable capacitances, resistances, and/or inductances if they are used. There are *N* nodes in any problem and vector ***b*** will require a length of 2•*N* to accommodate the two *B* constants (*BVi* and *BIi*) needed per circuit node. The actual number of nodes *N* in the problem can be much larger than one might think due to phantom nodes AND the requirement to break up transmission lines into smaller blocks to provide the necessary time resolution needed for the problem.

The first and last pairs of the ***b*** vector are unique as there are no prior (*i*-1) or following (*i*+1) values, respectively. Screamer treats these vector elements separately.

## Legacy Screamer Memory Conservation

Screamer minimized the need for computer memory by placing ALL circuit information needed for the solver (coefficients, etc.) in a single, long vector. This saved a great deal of memory by not using a matrix ***A*** representation having (2•*N*)2 elements and a separate ***b*** vector having 2•*N* elements. Accordingly, Screamer spends a great deal of effort to define and track a large number of indexes that are needed to populate the vector and solve the problem. This approach is very sparing of computer memory but very, very obscure. The details of the actual matrix to be solved and the solution mathematics are hidden from the

casual and not so casual observer. Understanding this minimally documented solver portion of Screamer was the product of extensive and painful code forensics. In many ways, this effort was the most difficult part of the problem.

## Legacy Screamer Matrix Solution Technique

The matrix solution to the sparse tri-diagonal matrix described above has significant mathematical problems. Many of the diagonal elements will always have a value of zero due to the large number of phantom nodes. See Eqs. 3.13 – 3.16 and Eq. 3.17. A matrix having diagonal values that are zero cannot be solved with the usual methods due to divide by zero problems in the solver algorithms.

*k k k*

*I*

*k k*

= 2*BV*

(3.15)

*i*1  2*AViVi*  *Ii*

* *IBl*,*i i*

*k k k k*

(3.16)

*Vi*  2*AIiIi*  *Vi*+1 = 2*BIi*

Kiefer & Widner2 addressed this problem by inverting the order of the current and voltage equations in the ***A*** matrix and the ***b*** vector (except for the first and last nodes – this is discussed later). This creates a pentadiagonal matrix in which the diagonal elements are guaranteed to be non-zero. Thus, the legacy solver in Screamer should properly be considered a very sparse, pentadiagonal solver. (There are tests in the Screamer solver to make sure that the first node diagonal coefficients are non-zero in all branches.) In the equations below we can see that each equation has two elements either following or preceding the new diagonal element. The example below is the inverted populated matrix from the End Branch example. Only the second and third rows are flipped in this case. In this case the diagonal elements can never be zero.

1

*AV*1

.5 0 0 0 0 0 0 0 0 *V*1

1 1

1

1

*BV*1

1

 *AI*1 .5 0 0 0 0 0 0 0 *I*1

1 1

*BI*1

1

0 0 1 2*AI* 2 1 0 0 1 0 0

1

*V*2 2*BI* 2

1 1

0 1 2*AV* 2 1 0 0 0 0 0 0

1

*I*2 2*BV* 2

1 1

0 0 0 .5 2*AV* 3 .5 0 0 0 0

1

*V*3 *BV*3

1 1

=

0 0 0 0 .5 2*AI* 3

0 0 0 0 *I*3

*BI*3

0 0 1 0 0 0 1 0 0 0 *V* 0

2

1

1

2 2

0 0 0 0 0 0 .5 *AI*1 .5 0 *I*1

2 2

0 0 0 0 0 0 0 .5 *AV*2 .5 *V*2

2 2

0 0 0 0 0 0 0 0 .5 *AI*2 *I*2

*BI*1

1

*BV*2

1

*BI*2

(4.3)

#### Matrix Solution Problems

This matrix still has problems with a solution. The first AI coefficient (second row) of all branches is nearly always zero as it often represents a phantom RLseries. In all top branches and end branches, the first *AI* coefficient is ALWAYS zero because branches always start with a null RLseries element. In the case of the first *AI* constant in a branch having a value of zero, Screamer places a small positive value in that matrix element. This arbitrary placement of a finite number in this matrix element can give bad results. While it was done for a good mathematical reason, it may create problems in the real world. This is the sole questionable assumption made in the legacy Screamer sparse-matrix solver.

The last *AI* coefficient value in any branch is always identically one and the prior matrix element in the row is always zero. This is because the final *AI* element is ALWAYS from a phantom RLseries block that is dangling. There can be NO current from the final node in any branch. Thus, the final row in a branch in matrix ***A*** multiplied by the ***x*** vector always equals the final value of *BI* in the ***b*** vector. This final value is ALWAYS zero. Thus, the final voltage equation in any *kth* branch always gives *Iki* = 0.

#### Screamer Legacy Solver Conclusion

The legacy Screamer sparse-matrix solver was fully optimized when it was written 30-years ago and no faster or more compact sparse-matrix solver exists even today to solve this particular problem. The solver remains impressively fast; scales as *N*; and is very stable.

The solver retains a single mathematical problem – the coefficient *AIk1* located in the second row of all branches is usually zero and Screamer treats this significant solver issue by inserting a small value (1X10-6) instead of zero. It is not clear that this particular number is optimum. The present Screamer solver uses double precision real mathematics (64 bit) and the solver could easily handle much smaller numbers without difficulty.

Sample matrix populations are presented in Appendices 1 & 2.

## Legacy Screamer Limitation on Branches in Branches

Screamer was deliberately set up to prohibit or preclude branches in branches. It was relatively simple to get Screamer to read in the circuit contents of higher-level branches. This involved changing some limits in the setup header files. It was not so easy to get Screamer to build the connections to the appropriate branches. Dr. Widner created a very obscure but very deliberate code sequence that limited Screamer to L2 branches. That section of the code is found in the *start\_run.f* subroutine. This is the section of the code that populates the data arrays that will then be used to populate the problem matrix. As part of this routine, the branch type and exit branch location information is embedded into a data array. The lines of code are given below.

c

c c Set the node for a variable element and the current number of nodes in c this branch.

c

if (ibt .ne. outputreq) then nvar\_node = nr(ib) + 1

nrx = nr(ib) + na

nr(ib) = nrx end if

c

c If we have a branch exiting this block, increment the exit

c branch counter, then set the circuit element index and the branch c exit type for this exit branch.

c

if (ibr .gt. 0) then

icb = icb + 1 indexb(1,icb) = nrx indexb(2,icb) = ibr

end if

In this piece of code the subroutine is iterating through the branches (*ibth* branch). Here the

code increments the number of nodes (circuit nodes) from the prior node count if a circuit block is seen. In this case, *ibt* is the circuit block type. The variable *na* is the number of nodes that were added to the branch node count due to the prior added circuit element. This number *na* is either 1 for an RCground or 2 for an RLseries element. (More complex blocks are ignored in this discussion.) The relevant thing to note here is that the new number of nodes *nrx* is defined and then replaces the old value of *nr(ib)*. The variable *nr(ib)* is the running node count for the branch *ib* in which one is presently located. Thus, after the first bit of code one has *nrx* that is the present node location *in that branch*.

The second bit of code asks if that node has a branch exiting from it. If *ibr* > 0 there is an exiting branch. The node location *nrx* is passed to the *indexb* matrix as the location and type of branch. The critical thing to note is that there is no allowance for exiting branches except from the main branch. If one blindly writes a run deck with L3 branches (these are branches exiting from the L2 branches) one sees that the *nrx* node count starts from the beginning of the L2 branch (at zero) not relative to the total node count. This means that the location of the connections is wrong. If one tries to build an L3 branch structure, the branch connectors can appear anywhere on the main branch or, in pathological cases, in the wrong place in the some earlier L2 branch. This causes the code to give totally incorrect results or, in some cases, to crash.

# Screamer V4.0

The impressively fast personal computers that are commonly available today also sport impressive memory sizes. It is common to see a personal computer, with > 8 GB of memory, approaching a teraflop of processing power. Furthermore, personal computer capabilities are continuing to increase very rapidly. We anticipate that 64-GB DRAM will be the default memory level by 2016 and that the development of > 8-core CPUs will result in multi- teraflop speeds. This continuing improvement in processing capability suggested that we revisit the Screamer matrix-solver algorithm and convert from a circuit-limiting, very sparse matrix solver to a more general matrix solver - even if, in the short term, there was a real increase in the computational time needed for typical problems. When we started this effort we did not know what the slow down factor would be.

## Screamer 4.0 Goals

Our goals for Screamer V4.0 are:

1. To implement a matrix solver that fundamentally allows branches in branches,
2. To have the problem matrix ***A*** and the problem vector ***b*** explicitly defined and directly traceable to the mathematical derivation of the problem,
3. To be able to diagnose the matrix elements for all population problems,
4. To have the matrix solver in a separate subroutine connected to Screamer only through passed variables – no common blocks, and
5. To do all of this with a minimal reduction in performance.

We also wanted to use this opportunity to make the matrix solver portion of Screamer more transparent and better documented (commented). This is critical for future support. All we would have to do is to change a single subroutine if a new matrix solver were needed at some time in the future.

## The Screamer V4.0 Solver

In this section we consider the implementation of the fast, direct linear solver developed for Screamer 4.0. The coefficient matrix in the resulting linear system is sparse and cannot be efficiently stored in the band formats adopted in LINPACK and LAPACK. Our attempt of using direct band solvers from the LAPACK package based on the band storage formats resulted in significant lost of efficiency in comparison with the existing solver adopted in previous Screamer releases on several benchmark problems. On some of the problems the CPU times increase by factors much more than 10. This forced us to adopt a sparse direct solver.

We choose to develop our own sparse direct solver instead of using existing sparse solvers such as PARDISO solver (http://www.pardiso−project.org) for two reasons: first, the sparse matrix storage format adopted in the Screamer is quite different from the standard sparse formats such as COO, CSR, CSC (see e.g. <http://en.wikipedia.org/wiki/Sparse>matrix) and so on; and, second, is the simplicity of the distribution pattern of entries in the coefficient matrices produced in Screamer. The developed solver exhibits the efficiency of the standard sparse solvers, i.e. the number of multiplications in the proposed solver is *O(N)*, where *N* is the number of non-zero entries in the sparse matrix. It also has a very transparent programming implementation due to the efficient storage format and a simple, repeating sparse-matrix structure. Next we describe the details of the developed sparse direct solver.

We start with the equations for the voltages

*n*,*k*

*Vi*

and currents

*n*,*k*

*I i*

, for every node

*i* = 1, …, *nk* in every branch *k* = 1, 2, …., *nb* at the *nth* time step that are given in Eqs. 3.11 &

3.12. For convenience we drop index *n* in this section. Here, the index *k* = 1 indicates the main branch, *nk* presents the number of nodes in the *kth* branch, and *nb* is the total number of branches. We restate the current and voltage equations derived in Section 3 below.

*k k k k k k*

*Ii*1  *AVi Vi*  0.5*Ii*  0.5*Ib l*,*i* = *BVi*

(3.13)

*k k k k k*

*Vi*  *AIi Ii*  0.5*Vi*+1 = *BIi*

(3.14)

where *i* = 1, …, *nk*-1 and *k* = 1, …, *nb*. The (end) branch connection conditions (Eqs. 3.19 &

3.20) in this case can be written as

*k l*

*Vb l* = *V*1

(3.19)

*k l*

*Ibl* = *I*1

(3.20)

Here, the index

*k*

*b l* indicates the node in the *kth* branch with the connection to the *lth* branch.

*k k*

We assume that

*b l* < *bm*

if *l* < *m*. If there is no connection to a branch at the *ith* node then the

last term in the left hand side of the first equation must be dropped. It is also assumed that a branch connected to another branch (not the main branch) cannot have any connections to other branches. We also list the branches connected to the branches after the branches

connected to the main branch and in the order of the branches they connected to, i.e.

*nl* < *nl*

if a branch

1 2

*nl*1 connected to a branch

*nk* 1 and the branch

*nl*2 connected to a branch

*nk* 2 with

0 < *nk* < *nk*

1 2 .

Unfortunately, the diagonal terms

*k*

*AVi*

and

*k*

*AIi*

in this system are often zero or very small.

This causes the need for partial or full pivoting in the standard direct solver algorithms such as LU factorization. This results in a significant efficiency loss on the forward factorization step. To avoid the pivoting procedure and to reduce the number of multiplications in the forward factorization step, we rewrite Eqs. 3.13 & 3.14 in the form

*k k k k k*

(3.16)

*Vi*  2*AIi Ii*  *Vi*+1 = 2*BIi*

*k k k k l k*

*I*

(3.15)

*i*1  2*AVi Vi*  *Ii*  *I*1 = 2*BVi*

Next we introduce the unknown solution vector ***x*** in the form

1 1 1 1

*nb nb*

*nb nb*

*nb nb T*

***x*** = *V*1 , *I* 1,...,*Vn* , *In* ,...,*V*1

1 1

, *I* 1 ,...,*Vn* , *In*

and the right-hand-side vector ***b*** as

1 1 1 1

*nb nb*

*nb nb T*

***b*** = 2*BI*1 , 2*BV*1 ,...,2*BIn* 1, 2*BVn* 1,...,2*BI*1 , 2*BV*1

,...,2*BIn nb*, 2*BVn nb*

. The resulting

linear system ***Ax*** = ***b*** can be presented in the following block form

1

1

*A* ... ... ... *V*1 *b*1

2 1

### ... *A*

... ... *I* 1

...

### ... ... ... ...

*nb*1

### ... ... *A* ...

... ... ... *Anb*

...

*nb*

*V*

*n*

*nb*

*nb*

*In*

*nb*

### = ...

...

*bnb*

(5.1)

In this system an *Ak* block corresponds to the *kth* branch. *Ak*, in turn, also has two rows of block structure corresponding to the nodes in the *kth* branch. In the system above we skipped the indication of the branch connections in the off diagonal elements but we will consider

them later. We also use the notation

*b* 2*k l*  1 for the first row in the *lth* block.

Then the *ith* and (*i* + 1)*st* 2-rows block of *Ak* can be presented in the form below.

... 1 *ak ak* 0 0 ... 0 ...

2*i*1,2*i* 2*i*1,2*i*+1

... 0 1 *ak* 0 0 ... *ak* ...

2*i*,2*i*+1

*k*

2*i*,2*bk* 1

... 0 0 1 2*AIi*+1 1 ... 0 ...

*k*

... 0  1 2*AVi*+1 1 0 ... ... ...

(5.2)

In this matrix the *ith* block has already been transformed into row-reduced echelon form and we consider the details of the transformation of the (*i* + 1)*st* block. For convenience, in this

*ak* , *l* = 1, ..., *mi*,*k mi*,*k*

matrix we indicate just one branch connection term

2*i*,2*bk l*

*j*,*l* where

*j*,*l*

is the

number of branches *kj* connected to the branch *kl* with *kj* < *k* < *kl*. It also includes the connection of the *lth* branch to the *kth* branch at the *vth* block, if *v* < *i*. It can be seen that the transformation of the (*i* + 1)*st* block of the *kth* branch to the reduced echelon form requires the elimination of two entries in the positions (2*i* + 2, 2*i*) and (2*i* + 2, 2*i* + 1) in the matrix block corresponding to the *kth* branch. The first step in the process requires only additions.

*ak* = *ak* + 2*AV k* ,

2*i*+2,2*i*+1

2*i*,2*i*+1

*i*+1

*ak* = *ak* + *ak* , *l* = 1, ..., *mi*,*k* ,

2*i*+2,2*b k l*

*k k*

*b*

2*i*,2*bk l*

*k*

2*i*+2,2*b k l*

*j*,*l*

(5.3)

2*i*+2 = *b* 2*i* + *b* 2*i*+2 .

Note that the second term in the second equation is almost always zero. The second elimination step requires just two multiplications.

*ak* = 1 + *ak* 2*AV k* ,

2*i*+2, 2*i*+2

2*i*+2, 2*i*+1

*i*+1

*ak* = *ak* ,

2*i*+2, 2*i*+3 2*i*+2, 2*i*+1

*bk* = *bk*  *ak k*

(5.4)

2*i*+2

2*i*+2

2*i*+1, 2*i*+1 *b* 2*i* .

The scaling of the (2*i* + 2)*th* row results in additional

*mi*,*k* + 2 multiplications. This shows

that the forward elimination step for the (*i* + 1)*st* block of the *kth* branch requires multiplication operations.

*j*,*l*

*i*,*k*

*j*,*l*

*m* + 2

Next, we consider the forward Gaussian elimination step at the first node of a branch *kl*. We assume that the branch *kl* is connected to the ith block of the branch *k* (*k* < *kl*). Also we consider that all rows above the first row in the block corresponding to the *lth* branch have already been transformed into row-reduced echelon form. Then we can present the relevant rows of the matrix in the form below.

... 1 *ak ak* ... 0 ...

2*i*1,2*i* 2*i*1,2*i*+1

... 0 1 *ak* ... 0 ...

2*i*,2*i*+1

... 0 ... ... ... ...

... *al* 0 ... 0 *al* ...

2*bk* 11,2*i*1

2*bk* 11, 2*bk* 11

(5.5)

*al* = 1 *al* = 1

Here,

2*bk l*1, 2*i*1

and

2*bk l*1, 2*bk l*1

. If we denote the difference between these

column positions by

2*l* = 2*bk*  2*i* , then we need ξ*l* iterations of a 2-step process to

sequentially move the first non-zero entry to the right and eventually eliminate it. The first step in the process is:

*l*

*al* = *al ak* ,

2*bk l*1, 2 *j*

2*bk l*1, 2 *j*1

2 *j*1, 2 *j*

*al* = *al*  *al ak* ,

2*bk l*1, 2 *j*+1

2*bk l*1, 2 *j*+1

2*bk l*1, 2 *j*1

2 *j*1, 2 *j*+1

*bl* = *bl*  *al bk* .

2*bk l*1

2*bk l*1

2*bk l*1, 2 *j*1

2 *j*1

(5.6)

We also can note that the first term in the right hand side of the second equation is almost

always zero. The second step can be written as

*al* = *al*  *al ak* ,

2*bk l*1, 2 *j*+1

2*bk l*1, 2 *j*+1

2*bk l*1, 2 *j*

2 *j*, 2 *j*+1

*al* = *al*  *al ak* , ** = 1, ..., *mi*,*k* ,

2*bk l*1, 2*bk *

2*bk l*1, 2*bk *

2*bk l*1, 2 *j*

2 *j*, 2*bk *

*j*,**

*bl* = *bl*  *al k*

2*bk l*1

*k l*

2*bk l*1

2*b* 1, 2 *j b* 2 *j* , *j* = 1, ..., *bk*  1 .

*l*

(5.7)

If the total number of the equations in the matrix is *N*, the number of multiplications required

*k*=*nb*

*i* = *nk*

on the back substitution step can be estimated by

2

3 *N* +

*k*=0 *i*=1

*i*, *k*

*j*, *l*

*m*

. We can see that if

the number of branches remains fixed, both the forward and back substitution steps are approximately *O(N)*. So, the resulting algorithm presents a very efficient, direct linear solver.

In our description, we only allow connections of a branch (L2) to the main branch (L1) or a branch (L3) to branches (L2) connected to the main branch (L1). But, as a matter of fact, the constructed direct linear solver has no limitation on the number of embedded branch levels. Essentially, there is only one restrictions on the branch connections: If a branch *kl* connected to a branch *km* then the entry in the unknown vector corresponding to the first node of the branch *kl* must come after all entries in the unknown vector corresponding to nodes in the branch *km*.

## V4.0 Solver Implementation

We implemented the new solver routines in Screamer in two, very clear steps:

1. We carefully populated the solution matrix with the Screamer-generated matrix coefficients with a separate subroutine (*solver\_band\_matrix1.f*) and
2. We created a completely separate, stand-alone matrix solver subroutine (*solver\_mdgauss.f*)

– in parallel with the existing solver.

This approach allowed us to gain a very complete understanding of the detailed structure of the solution matrix that was generated by Screamer and to be able to directly compare potential new matrix solvers with the existing legacy solver.

## Screamer 4.0 Software Modifications for Branches in Branches

As discussed in Section 4.5, the legacy Screamer code *implicitly* blocks L3 branches in L2 branches in *start\_run.f*. This is done by never incrementing the total node counter as one moves to a new branch to populate the matrix components. The result is that branch coupling- coefficients were deliberately placed in the wrong location if branches were placed in branches (branches were called from branches other than the main branch). It was relatively straightforward to fix the code. The selection below causes the subroutine to generate the correct problem matrix ***A*** with the correct branch connectors.

c

c

c Set the node for a variable element and the current number of nodes in c this branch.

c

if (ibt .ne. outputreq) then nvar\_node = nr(ib) + 1 nrx = nr(ib) + na nr(ib) = nrx

end if

c

c If we have a branch exiting this block, increment the exit

c branch counter, then set the circuit element index and the branch c exit type for this exit branch.

c

if (ibr .gt. 0) then icb = icb + 1 nodecount = 0

do i=1,ib-1

nodecount = nodecount + nr(i) enddo

nnrx = nodecount + nrx indexb(1,icb) = nnrx indexb(2,icb) = ibr

end if

c

Here, we added a do loop that counts up the nodes in all prior branches and includes the node count up to the exiting branch in that branch. The loop in embedded inside another loop that will iterate over all of the *ibth* branches until the *nbth* (maximum branch) is reached. We need only count up the branches to the *nb*-1 branch because the final branch in the problem can

NEVER have an exiting branch (or it would not be the final branch). Tests done with this version of the code demonstrate that the ***A*** matrix is populated correctly.

While the fix to the subroutine is simple, it took many weeks of forensic analysis of the Screamer subroutines, the indices, and the code logic to understand the problem and to fully appreciate the subtle, deliberate cleverness of the method Dr. Widner used to limit Screamer to L2 branches. A more formal logic check might have permitted an unwary user in the distant future to enable branches in branches without the solver being able to handle that situation.

# The Existing Computational Issue Remains

The discussion in the sparse matrix solutions above describes the problem with initial rows in branches that have values of zero for the diagonal elements. The first two rows of the main branch could have their order flipped (like the following rows) as the *AI1* and *AV1* (*k* = 1) could be zero. Screamer does not flip the first two rows in the main branch (*k* = 1) but checks for zeros and puts in a very small coefficient (1X10-6) if a coefficient is zero. The first two rows of an end or side branch cannot be flipped as the element to the right of the diagonal of the first row of the *lth* branch is always zero. (Rather than treat the main branch separately from other branches, Screamer does not flip the first two rows in ANY branch.) But the problem is that all top branches and end branches start with a phantom RLseries block in which the *AI1* of the first block is always zero. So either we don’t flip the first two rows of an end or top branch and have the second diagonal element ALWAYS zero in the voltage equation or we flip the rows and have the second diagonal zero from the current equation. There is no obvious reason not to flip the last two rows of all branches. Screamer does not like the value of G or C in the last block to be zero!

# Conclusion

We have completed a complete rewrite of the Screamer matrix solver routine. This new speed-optimized solver is only 2X slower than the legacy Screamer solver but is capable of solving problem matrices that are not sparse. While implementing the matrix solver we took the legacy Screamer vector containing the matrix coefficients and used them to populate the ***A*** matrix in a clear and transparent fashion. This allows for a simple diagnostic analysis of the problem matrix and a direct comparison with the problem mathematics.

The new, full-matrix solver allows us to attack more complex problems that populate the ***A*** matrix with many more off pentadiagonal components. We refer to this in Screamer as allowing “branches-in-branches”. Version 4.1 of Screamer will now allow unlimited Level-3 (L3) branches. This means that in addition to branches (L2) off of the main branch (L1), we allow branches (L3) off of L2 branches. This single change allows for vastly more complex machine topologies to be modeled. Screamer V4.1 is now fully operational on Macintosh/LINUX/UNIX/Windows platforms.

# References

1. J. C. Bowers & S. R. Sedore, Sceptre, A Computer Program for Circuit and System Analysis, Englewood Cliffs, NJ, Prentis Hall 1971.
2. M. L. Kiefer & M. M. Widner, IEEE Pulsed Power Conference, 1985

# Appendix 1 Populating a Simple Matrix Without Branches

Let’s use a very simple run deck without branches and manually put ACTUAL values in each of the array elements. The current and voltage equations are taken directly from Eqs. 3.13 – 3.16.

*k k k*

*I*

*k k*

= 2*BV*

(3.15)

*i*1  2*AViVi*  *Ii*

* *IBl*,*i i*

*k k k k*

(3.16)

*Vi*  2*AIiIi*  *Vi*+1 = 2*BIi*

where

*AVi* = 0.5*Gi* 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Ci* | *o o o*  *and BV* = 0.5 *I*  *I*  *I*  | | *Ci* | *o*   0.5*G V* |
| *t*  *Li* | *i i*1 *i Bi*  *o o Li*  *and BI* = 0.5 *V*  *V*  |  | *t i i*  *o*  0.5*R I* | |
| *t* | *i i i*+1 *t* |  |  | *i i* (3.10) |

*AIi* = 0.5*Ri* 

In our case the branch current in the current equation is zero because this example has no branches.

## A1.1 Sample Run Deck

Our run deck, no\_branch\_test.txt, is listed below.

No Branch example

!

! 2014-06-12 RBS

!

! This is a simple LCR circuit with no branches but several elements

!

Time-step 0.25e-9 Resolution-time 2e-9 End-time 1e-6 Number-prints 5 Execute-cycles all Grids no

Echo-setup no Max-points 3001

!

!Start circuit definition

!

! Main Branch

! BRANCH

RCG 1e+12 1000e-9

Initial VC1 80e3 UFO VC1

$Voltage(V)

!

RLseries 0.0 15e-9

!

RLseries 0.0 35e-9 UFO IR2

$I(A)

!

!Load to ground

!

RCground 0.1 0.0

!

! End circuit

## A1.2 Calculate the Coefficients

The values for the block elements for the nodes are provided. We can now use those values to calculate the exact values of the *AVi*, *AIi*, *BVi*, and *BIi* coefficients for the first time step in which all of the old values are 0. In this case rht = 4X109 s-1.

RCG (element #1)

AV1 = (0.5G1 + rht\*C1) = 4X103 AI1 = 0

BV1 = 0.5\*(-Io1) + (rht\*C1 + 0.5\*G1)\*Vo1 = 4X109 \* 10-6 \* 8X104 = 3.2X108 BI1 = 0.5\*(Vo1-Vo2) = 0.5\*80,000 = 4X104

RLS (element #2)

AV2 = 0

AI2 = 0.5\*R2 + rht\*L2 = 60 BV2 = 0.5\*(Io1-Io2) = 0

BI2 = 0.5\*(Vo2-Vo3) + (rht\*L2 - 0.5\*R2)\*Io2 = 0 AV3 = 0

AI3 = 0

BV3 = 0.5\*(I02-Io3) = 0 BI3 = 0.5\*(Vo3-Vo4) = 0

RLS (element #3)

AV4 = 0

AI4 = 0.5\*R4 + rht\*L4 = 140 BV4 = 0.5\*(Io3-Io4) = 0

BI4 = 0.5\*(Vo4-Vo5) + (rht\*L4 - 0.5\*R4)\*Io4 = 0 AV5 = 0

AI5 = 0

BV5 = 0.5\*(I04-Io5) = 0 BI5 = 0.5\*(Vo5-Vo6) = 0

RCG (element #4)

AV6 = (0.5G6 + rht\*C6) = 0.5\*10 = 5 AI6 = 0

BV6 = 0.5\*(Io5-Io6) + (rht\*C6 + 0.5\*G6)\*Vo6 = 0 BI6 = 0.5\*(Vo6) = 0

## A1.3 Populate the Matrix

The array below is an example of how the ***A*** matrix is initially populated. We will be flipping the rows in pairs, except for the first and last rows just as was done in Eq. 26. The first and last rows are not scaled by 2X. This includes the ***b*** vector values. You can see that the ***A*** matrix is pentadiagonal where the diagonal elements are the *AIi* and *AVi* content and the *BI* and *BV* are in the output vector.

1

*AV*1

0.5 0 0 0 0 0 0 0 0 0 0 *V*1

1 1

1

1

*BV*1

1

0.5 *AI*1 0.5 0 0 0 0 0 0 0 0 0 *I*1

1 1

*BI*1

1

0 0 1 2*AI* 2 1 0 0 0 0 0 0 0 *V*2

1 1

2*BV* 2

1

0 1 2*AV* 2 1 0 0 0 0 0 0 0 0

1

*I*2 2*BI* 2

1 1

0 0 0 0 1 2*AI* 3 1 0 0 0 0 0 *V*3

1 1

2*BV* 3

1

0 0 0 1 2*AV* 3 1 0 0 0 0 0 0

1

*I*3 2*BI* 3

1 1

=

0 0 0 0 0 0 1 2*AI* 4 1 0 0 0 *V*4

1 1

2*BV* 4

1

0 0 0 0 0 1 2*AV* 4 1 0 0 0 0

1

*I*4 2*BI* 4

1 1

0 0 0 0 0 0 0 0 1 2*AI* 5 1 0 *V*5

1 1

2*BV* 5

1

0 0 0 0 0 0 0 1 2*AV* 5

1 0 0 *I*5

1 1

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.5 | *AV*6 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  |

0.5 *V*6

*AI*6 *I*6

1

2*BI* 5

1

*BV*6

1

*BI*6

(A1-1)

For our case many of the *AIi* and *AVi* values are zero because they are for phantom elements where G, C, R, and L are zero. They will nearly ALWAYS be 0. If you fill the nodes for the first time step you get the matrix below. Note: the initial voltage conditions are located in *BIi*. In the case of the run deck branch\_test.txt on all of the *BIi* in the first time step only the first element is non zero, here *BI1* = 4X104. There is also an instantaneous current in *BVi* for the first time step. In this case for branch\_test.txt *BV1* = 3.2X108.

1

*V*1

1

*I*1

*V* 1 3.2*x*10

8

2

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 4000 | .5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| .5 | 0 | .5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 120 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 280 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |  1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  .5 | 5 | .5 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  .5 | 0 |

1 40,000

*I*2 0

1

*V*3 0

1 0

*I*3 0

=

*V* 1 0

4 0

1

*I*4 0

1 0

*V*5 0

1 0

*I*

5

1

*V*6

1

*I*6

(A1-2)

## A1.4 Screamer Generated Matrix

The array that Screamer generates internally and *really solves* is shown below in a direct print out from the code. We have implemented a switch that allows us to print out the elements of the matrix for the first time step. This ensures that the population routines actually fill the matrix correctly.

***A*** matrix (from MTRX)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **4e3** | .5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| -.5 | **e-6** | .5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | **1** | -120 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | -1 | 0 | **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | **1** | 0 | -1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | -1 | 0 | **-1** | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | **1** | -280 | -1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | -1 | 0 | **1** | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **1** | 0 | -1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | 0 | **1** | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -.5 | **5** | -.5 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **1** |

(A1-3)

You can see that there are several differences between our analytic example and Screamer. Screamer has a floor on the value of 1X10-6 for the first value of *AI1* in a branch because the coefficient cannot be zero. The diagonal elements are in **bold**. Note, the first node and the last node of the problem are handled separately in the solver so the fact that they were not reversed in the actual internal matrix is not important. Note, Screamer sets the last *AIi* of the main branch to one and the prior element in the last row to 0. There is no current can flow through this phantom block so by setting these values Screamer always forces the final current of ANY branch to be 0.

***B*** vector (from RHS)

3.2X108

4X104

0

0

0

0

0

0

0

0

0

0

The first two elements of the vector ***B*** are exactly as calculated in our test matrix. Recall that the first two rows are not scaled by 2. The final *AIi* element is always zero.

# Appendix 2 Populating a Matrix with a Single End Branch

Let’s use the run deck branch\_end\_test.txt that has an end branch and put ACTUAL values in each of the array elements. The current and voltage equations are taken directly from Eqs.

3.13 – 3.16. Recall that we are swapping the voltage and current equations for all but the first and last row of both branches.

*k k k k k*

*Ii*1  *AViVi*  0.5*Ii*  0.5*IBl*,*i* = *BVi*

(3.13)

*k k k k*

*Vi*  *AIiIi*  0.5*Vi*+1 = *BIi*

(3.14)

*k k k*

*I*

*k k*

= 2*BV*

(3.15)

*i*1  2*AViVi*  *Ii*

* *IBl*,*i i*

*k k k k k*

(3.16)

*Vi*  2*AIi Ii*  *Vi*+1 = 2*BIi*

Where the coefficients were defined earlier in Eq. 3.10.

*AVi*

= 0.5*Gi* 

*o*

*Gi Vi*

|  |  |  |  |
| --- | --- | --- | --- |
| *Ci* | *o o o*  *and BV* = 0.5 *I*  *I*  *I*  | | *Ci*  0.5 |
| *t*  *Li* | *i i*1 *i Bi*  *o o Li*  *and BI* = 0.5 *V*  *V*  |  | *t*  *o*  0.5*R I* |
| *t* | *i i i*+1 *t* |  | *i i* |

*AIi*

= 0.5*Ri* 

(3.10)

## A2.1 Sample Run Deck

The run deck branch\_end\_test.txt is the following:

End Branch test

!

! End Branch Test

!

Time-step 0.25e-9 Resolution-time 2e-9 End-time 1e-6 Number-prints 5 Execute-cycles all Grids no

Echo-setup no Max-points 1001

!

!Start circuit definition

!

! Main Branch - (Branch #1)

! BRANCH

RCG 1e+12 1e-6

Initial VC1 80e3 UFO VC1

$Voltage(V)

!

RLseries 0.0 20e-9

!

RCground 2e+6 0.0 EndBranch

! Branch #2 location

!

RLseries 0.0 25e-9

!

RCground 0.1 0.0

!

!End Main Branch

!

!Level 2 End Branches

!

!Branch #2 Branch RLseries 0.1 0.0

RCground 0.25 0.0

! End circuit

## A2.2 Calculate the Coefficients

We start with the clear formulation from the branchless analysis to get the following node values. We can now use those values to calculate the exact values of the *AVi*, *AIi*, *BVi*, and *BIi* coefficients for the first time step in which all of the old values for current and voltage are 0. In this case, rht = 4X109 s-1.

RCG (element #1)

AV1 = (0.5G1 + rht\*C1) = 4X103

AI1 = 0.5\*R1 + rht\*L1 = 0 (always)

BV1 = 0.5\*(-Io1) + (rht\*C1 + 0.5\*G1)\*Vo1 = 4X109 \* 10-6 \* 8X104 = 3.2X108 BI1 = 0.5\*(Vo1-Vo2) + (rht\*L1 - 0.5\*R1)\*Io1 = 0.5\*80,000 = 4X104

RLS (element #2) AV2 = 0 (always)

AI2 = 0.5\*R2 + rht\*L2 = (.5\*0 + 4X109\*20X10-9) = 80

BV2 = 0.5\*(Io1-Io2) + (rht\*C2 + 0.5\*G2)\*Vo2 = 0 BI2 = 0.5\*(Vo2-Vo3) + (rht\*L2 - 0.5\*R2)\*Io2 = 0

AV3 = 0 (always) AI3 = 0 (always)

BV3 = 0.5\*(I02-Io3) = 0

BI3 = 0.5\*(Vo3-Vo4) + (rht\*L3 - 0.5\*R3)\*Io3 = 0

RCG (element #3)

AV4 = (0.5G4 + rht\*C4) = 2.5X10-7

AI4 = 0 (always)

BV4 = 0.5\*(-Io1) + (rht\*C1 + 0.5\*G1)\*Vo1 = 0 BI4 = 0.5\*(Vo4-Vo5) + (rht\*L4 - 0.5\*R4)\*Io4 = 0

RLS (element #4) AV5 = 0 (always)

AI5 = 0.5\*R5 + rht\*L5 = (.5\*0 + 4X109 \*25e-9)= 100 BV5 = 0.5\*(Io3-Io4) = 0

BI5 = 0.5\*(Vo5-Vo6) + (rht\*L5 - 0.5\*R5)\*Io5 = 0

AV6 = 0 (always) AI6 = 0 (always)

BV6 = 0.5\*(I04-Io5) = 0 BI6 = 0.5\*(Vo5-Vo6) = 0

RCG (element #5)

AV7 = (0.5G7 + rht\*C7) = 5

AI7 = 0 (always)

BV7 = 0.5\*(Io5-Io6) + (rht\*C6 + 0.5\*G6)\*Vo6 = 0 BI7 = 0.5\*(Vo6) + (rht\*L7 - 0.5\*R7)\*Io7 = 0

End Branch

RLS (Dummy RLS)

AV1 = 0 (Voltage Equation) AI1 = 0 (always)

BV1 = 0.5\*(I01-Io2) = 0 BI1 = 0.5\*(Vo2-Vo3) = 0

RLS (element #1) AV2 = 0 (always)

AI2 = 0.5\*R2 + rht\*L2 = (.5\*.1 + 4X109 \*0)= .05 BV2 = 0.5\*(Io1-Io2) = 0

BI2 = 0.5\*(Vo4-Vo5) + (rht\*L4 - 0.5\*R4)\*Io4 = 0

AV3 = 0 (always) AI3 = 0 (always)

BV3 = 0.5\*(I04-Io5) = 0 BI3 = 0.5\*(Vo5-Vo6) = 0

RCG (element #2)

AV4 = (0.5G6 + rht\*C6) = 2

AI4 = 0 (always)

BV4 = 0.5\*(Io5-Io6) + (rht\*C6 + 0.5\*G6)\*Vo6 = 0 BI4 = 0.5\*(Vo6) = 0

## A2.3 Populate the Matrix

The array shown in Eq. A2.1 below is an example of how the ***A*** matrix is populated directly from Eqs. 3.13 – 3.16. You can see that the ***A*** matrix is pentadiagonal where the diagonal elements are the *AIi* and *AVi* content and the *BI* and *BV* are in the output vector.

0 *V*1

1

1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

|  |  |  |
| --- | --- | --- |
| 1  *AV*1 | .5 | 0 |
|  | 1  *AI*1 | .5 |

0 *I*1

1 1

|  |  |  |
| --- | --- | --- |
| 0 | 0 | 1 |
| 0 | 1 | 1  2*AV* 2 |

1

*BV*1

1

*BI*1

1

2*AI* 2

1

1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2*AI* 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

1

1

*V*2 2*BI* 2

1 1

*I*2 2*BV* 2

1 1

*V*3 2*BI* 3

1 1

2*AV* 3

1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2*AI* 4 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1  2*AI* 5 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

0 0 1

1

*I*3 2*BV* 3

1 1

*V*4 2*BI* 4

1 1

0 1 2*AV* 4

1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |  |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

2*AV* 5

1 0 0 0 0 0 0 0 0 0 0 0 0

1

*I*4 2*BV* 4

1 1

*V*5 2*BI* 5

1 1

*I*5 2*BV* 5

1 1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 0 1 2*AI* 6 | | | | | | | | | | | | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1  0 1 2*AV* 6 1 | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | .5 | 1  *AV*7 | .5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | .5 | 1  *AI*7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*V*6 2*BI* 6

=

1 1

*I*6 2*BV* 6

1 1

*V*7 *BV*7

1 1

*I*7 *BI*7

0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 *V* 0

2

1

2

2 2

0 0 0 0 0 0 0 0 0 0 0 0 0 0 .5 *AI*1 .5 0 0 0 0 0 *I*1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2*AV* |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*BI*1

2

2



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2*AI* | 2 | 1 | 0 | 0 | 0 |
| 1 |  | 0 | 0 | 0 | 0 |

2

2

2

2*AI* 3 1 0

|  |  |  |
| --- | --- | --- |
| 0 |  | 1 |
| 1  0 |  | 2*AV*  0 |
| 0 |  | 0 |

2

3 1 0 0

2

.5 *AV*4 .5

2

0 .5 *AI*4

2 2*BI* 2

2 2

*V*

2 2*BV* 2

*I*

2 2

2 2*BI* 3

*V*

3 2

2 2*BV* 3

*I*

3 2

2 *BV*4

*V*

4 2

2 *BI*4

*I*

4

(A2.1)

For our case, many of the *AIi* and *AVi* values are zero because they are for phantom elements where *G*, *C*, *R*, and *L* are zero. If you fill the nodes for the first time step you get the matrix below. Note: the initial voltage conditions are located in *BIi*. In the case of the run deck, branch\_end\_test.txt, on all of the *BIi* in the first time step only the first element is non zero, here *BI1* = 4X104. There is also an instantaneous current in *BVi* for the first time step. In this case for branch\_end\_test.txt *BV1* = 3.2X108.

1

(A2.2)

*V*1

1

*I* 1

1

*V*2

1

*I* 2

1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 4000 | .5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| .5 | 0 | .5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 160 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 |  1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |  1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |  1 | 5*e*7 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |  200 |  1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |  1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |  1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  .5 | 5 | .5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | .5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  .5 | 0 | .5 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |  .1 |  1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | .5 | 2 | .5 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | .5 | 0 |

*V*3 3.2*x*108

*I* 1 40,000

3

1 0

*V*4 0

1 0

*I*

4 0

*V* 1 0

0

5

1

*I* 5 0

1 0

*V*6 0

=

1

0

*I* 6 0

*V* 1 0

7 0

*I* 1 0

7

2 0

*V*1 0

2 0

*I* 1 0

2 0

*V*

2 0

2

*I* 2

2

*V*3

2

*I* 3

2

*V*4

2

*I* 4

## A2.4 Screamer Generated Matrix

The array that Screamer generates internally and *really solves* for the case of a single end branch is shown below in a direct print out from the code. All of the diagonal elements are in **bold**.

***A*** matrix (from MTRX)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **4e3** | .5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| -.5 | **e-6** | .5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | **1** | -160 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | -1 | 0 | **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | **1** | 0 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | -1 | 0 | **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | **1** | 0 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | -1 | 5e-7 | **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **1** | -200 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | 0 | **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **1** | 0 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | 0 | **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -.5 | **5** | .5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -.5 | **e-6** | .5 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **1** | -.1 | -1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | 0 | **1** | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **1** | 0 | -1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | 0 | **1** | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -.5 | **2** | .5 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **1** |

Compare the Screamer-generated matrix to the matrix we generated in the prior section. You can see that there are several differences between our analytic example in the prior section and the Screamer output. Screamer has a floor of 1X10-6 for the first value of *AI1* in a branch because the coefficient cannot be zero. (This happens in the first *AI1* of the end branch as well as the main branch.) Again note, the first node and the last node of each branch are handled separately in the solver - they were not reversed, multiplied by 2, or the voltage equation multiplied by -1. Note, Screamer sets the last *AVi* of both branches to one and the prior element in the last row to 0. There is no current can flow through this phantom block so by setting these values Screamer always forces the current from the final node of all branches to be 0.

***B*** vector (from RHS)

3.2X108

4X104

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

The first two elements of the vector ***B*** are exactly as calculated in our test matrix. Recall that the first two rows are not scaled by 2. The last *BIi* of each branch is always zero.

# Appendix 3 Gaussian Solver Subroutine

SUBROUTINE solvermdgaussH(am,rhs,nk,max\_bb,nadd\_array,nb,nr) C

C This SUBROUTINE solvermdgaussH solves the linear system using modified C Gaussian elimination procedure based on highly structured sparse

C matrix obtained on the previous step. The solution vector is in the C rhs\_band vector.

C C=======================================================================

|  |  |  |  |
| --- | --- | --- | --- |
| C | input: | am | - the array containing the nk by nk |
| C |  |  | coefficient matrix |
| C |  | rhs | - the vector containing the right hand |
| C |  |  | side in the system |
| C |  | nk | - the size of the system |
| C |  | max\_bb | - the leading dimension of array am |
| C |  | nadd\_array | - the branch connection location |
| C |  | nb | - the number of branches |
| C |  | nr | - the number on nodes in the branches |
| C | output: |  |  |
| C |  | rhs | - the solution vector. |

C=======================================================================

C Dr. Yury Gryazin, 06/19/2014, ISU, Pocatello, ID C Final version 11/10/2014

C

INTEGER nk, nb, nbm, nadd\_array(\*), jadd, nr(\*) REAL\*8 am(max\_bb,\*), rhs(\*), ATS, BTS, CTS CHARACTER\*5 NAME, NAME1

c DO I = 1,nb

c PRINT \*,' I=',I,nadd\_array(I)

c END DO

c read \*, kkkk nbm = nb - 1 DO ib = 1,nb

C

C The first node in the ib branch C

i = 2 \* nadd\_array(ib) + 1

ATS = 1.0/am(i,i) am(i,i+1) = am(i,i+1)\*ATS am(i,i ) = 1.0

am(i,i+2) = 0.0

DO ibb = ib+1, nb

j = 2 \* nadd\_array(ibb) + 2 am(i,j) = am(i,j)\*ATS

END DO

rhs(i) = rhs(i)\*ATS

BTS = 1.0/( am(i+1,i+1) - am(i+1,i)\*am(i,i+1) ) am(i+1,i+2) = am(i+1,i+2)\*BTS

am(i+1,i+1) = 1.0

DO ibb = ib+1, nb

j = 2 \* nadd\_array(ibb) + 2

am(i+1,j)= (am(i+1,j) - am(i+1,i)\*am(i,j) )\*BTS END DO

rhs(i+1)= (rhs(i+1) - am(i+1,i)\*rhs(i) )\*BTS DO ibb = ib+1, nb

ii = 2 \* nadd\_array(ibb) + 1

am(ii,i+1) = am(ii,i+1) - am(ii,i)\*am(i,i+1)

DO ibbb = ib+1, nb

j = 2 \* nadd\_array(ibbb) + 2 am(ii,j)= am(ii,j)- am(ii,i)\*am(i,j) END DO

rhs(ii) = rhs(ii) - am(ii,i)\*rhs(i) END DO

C

C The nodes from 2 to nr(ib)-1 C

nrm = nr(ib) - 1 DO n = 2,nrm

i = 2 \* nadd\_array(ib) + 2 \* n

im1 = i-1 im2 = i-2

c

c The first row (Voltage equation) is already in the upper diagonal c form so we are working only with the second equation in a block. c Current equation, second row. Nonzero elements are

c am(j,j+1),am(j,j+2) and possibly am(j,nb(ib)). c

ATS = am(i,i-1) - am(i-2,i-1)\*am(i,i-2)

BTS = 1.0/( am(i ,i ) - am(i-1,i)\*ATS ) am(i,i+1) = -am(i-1,i+1)\*ATS\*BTS

am(i,i ) = 1.0

DO ibb = ib+1, nb

j = 2 \* nadd\_array(ibb) + 2

am(i,j) = am(i,j) - am(i-2,j)\*am(i,i-2)

am(i,j) = (am(i,j) - am(i-1,j)\*ATS)\*BTS END DO

rhs(i) = rhs(i)-rhs(i-2)\*am(i,i-2)

rhs(i) = (rhs(i)-rhs(i-1)\*ATS)\*BTS

DO ibb = ib+1, nb

ii = 2 \* nadd\_array(ibb) + 1

ATS = am(ii,i-1)-am(i-2,i-1)\*am(ii,i-2)

am(ii,i ) = am(ii,i ) - am(i-1,i )\*ATS am(ii,i+1) = am(ii,i+1) - am(i-1,i+1)\*ATS

DO ibbb = ib+1, nb

j = 2 \* nadd\_array(ibbb) + 2

am(ii,j) = am(ii,j) - am(i-2,j)\*am(ii,i-2)

am(ii,j) = am(ii,j) - am(i-1,j)\*ATS END DO

rhs(ii) = rhs(ii) - rhs(i-2)\*am(ii,i-2) rhs(ii) = rhs(ii) - rhs(i-1)\*ATS

END DO END DO

C

C The last node in the ib branch C

i = nr(ib)\*2 + 2\*nadd\_array(ib) - 1 ATS = 1/(am(i,i)-am(i-1,i)\*am(i,i-1)) BTS = 1.0/am(i+1,i+1)

am(i ,i+1) = am(i,i+1)\*ATS am(i ,i ) = 1.0 am(i+1,i+1) = 1.0

DO ibb = ib+1, nb

j = 2 \* nadd\_array(ibb) + 2

am(i ,j) = ( am(i,j) - am(i-1,j)\*am(i,i-1) )\*ATS am(i+1,j) = am(i+1,j)\*BTS

END DO

rhs(i ) = ( rhs(i) - rhs(i-1)\*am(i,i-1) )\*ATS rhs(i+1) = rhs(i+1)\*BTS

DO ibb = ib+1, nb

ii = 2 \* nadd\_array(ibb) + 1

ATS = am(ii,i ) - am(i-1,i )\*am(ii,i-1) BTS = am(ii,i+1) - am(i ,i+1)\*ATS

DO ibbb = ib+1, nb

j = 2 \* nadd\_array(ibbb) + 2

am(ii,j) = am(ii,j) - am(i-1,j)\*am(ii,i-1) am(ii,j) = am(ii,j) - am(i ,j)\*ATS

am(ii,j) = am(ii,j) - am(i+1,j)\*BTS END DO

rhs(ii) = rhs(ii) - rhs(i-1)\*am(ii,i-1) rhs(ii) = rhs(ii) - rhs(i )\*ATS rhs(ii) = rhs(ii) - rhs(i+1)\*BTS

END DO END DO

ktime = 1

if (ktime .eq. 0) then

NAME ='UMTRX' NAME1='URHS'

OPEN(57,FILE=NAME)

DO I = 1,nk

WRITE(57,\*) 'I=',I WRITE(57,\*) (am(I,J),J=1,nk) ENDDO

CLOSE(57) OPEN(58,FILE=NAME1)

DO I = 1, nk

WRITE(58,\*) I,rhs(I) ENDDO

CLOSE(58)

PRINT \*, 'THE UPPER TRIANGULAR MATRIX AND CORRESPONDING RHS

&ARE STORED in UMTRX and URHS files.' STOP

end if

C

C Back substitution C

C

C The last branch C

ibr = nb

i = nr(ibr)\*2 + 2\*nadd\_array(ibr) - 1 rhs(i) = rhs(i) - am(i,i+1)\*rhs(i+1)

nrm = nr(ibr)-1

DO n = 1,nrm

i = 2 \* nadd\_array(ibr) + 2\*(nr(ibr)-n) rhs(i ) = rhs(i ) - am(i ,i+1)\*rhs(i+1)

rhs(i-1) = rhs(i-1) - am(i-1,i )\*rhs(i ) & - am(i-1,i+1)\*rhs(i+1)

END DO

C

C The branches from nb-1 to 1 C

DO ib = 1,nb-1 ibr = nb - ib

i = nr(ibr)\*2 + 2\*nadd\_array(ibr)

DO ibb = ibr+1, nb

j = 2 \* nadd\_array(ibb) + 2 rhs(i) = rhs(i) - am(i,j)\*rhs(j) END DO

rhs(i-1) = rhs(i-1) - am(i-1,i)\*rhs(i)

DO ibb = ibr+1, nb

j = 2 \* nadd\_array(ibb) + 2

rhs(i-1) = rhs(i-1) - am(i-1,j)\*rhs(j) END DO

nrm = nr(ibr) - 1

DO n = 1,nrm

i = 2 \* nadd\_array(ibr) + 2\*(nr(ibr)-n) rhs(i) = rhs(i) - am(i,i+1)\*rhs(i+1)

DO ibb = ibr+1, nb

j = 2 \* nadd\_array(ibb) + 2 rhs(i) = rhs(i) - am(i,j)\*rhs(j) END DO

rhs(i-1) = rhs(i-1) - am(i-1,i)\*rhs(i)

& - am(i-1,i+1)\*rhs(i+1)

DO ibb = ibr+1, nb

j = 2 \* nadd\_array(ibb) + 2

rhs(i-1) = rhs(i-1) - am(i-1,j)\*rhs(j) END DO

END DO END DO

RETURN END