Abstract

This paper describes the **limit matrix** calculations done in the ANP software developed by William Adams. A brief description of the algorithms used is given, and then a closer look at how the calculations handle a couple of examples is given.

1 Introduction

The ANP software provides a low-level library to perform ANP calculations, and a GUI to facilitate the development of complex networks. In the process of synthesis of alternatives, limit matrix calculations need to be performed. In the case of a sufficiently connected network, this calculation is relatively easy [1]. However, in the insufficiently connected network case (e.g. a hierarchy) the calculation is much more difficult. Let us here define what an insufficiently connected network is:

Insufficiently connected network This is a network which has sinks. Sinks are nodes which are not *with respect to nodes* for any comparisons. Or using the standard of with respect to nodes being from nodes, a sink is a node which connects to nobody (although he may connect from many). These nodes can be identified as columns in the super matrix which are completely zero.

In this case, larger and larger powers of the matrix simply vanish, and we are left with no information. One way to fix this is to replace those zero columns by the same column from the identity matrix (i.e. if column 7 of a 12x12 matrix is zero, replace it by column 7 of a 12x12 identity matrix). We call these columns **identity columns**. There are many other ways to remedy this situation, as we shall see shortly.

2 Technical Description

Here we shall get a complete look at the calculations, and how they are actually carried out in the c code. We have three classes of calculations:

Vargas's calculation I give this a separate section because it is neither out of date, nor current. I do not believe this calculation method to be necessary, however, it is included for completeness.

Out of date These are older calculation methods with many flaws in them. Although they given fun results from time to time, they are never to be trusted. You should only use these for play purposes.

[1] There is, however, the multiple unit eigenvalue case. Although this case should be handled in the same way as any other sufficiently connected network, there is some discussion about this. As a result, a more complicated formula is included in the software which implements Luis Vargas's adaptation of the standard formula for this case. This formula may be turned off by setting a global variable (or in limit matrix options in the GUI .

Current These calculation models work reasonably well in all instances. They should handle sufficiently connected networks, and hierarchies correctly. However these methods may disagree in the other case.

Let's look at how these methods work. For the purposes of these discussions, let:

W The matrix we are taking the limit of.

n The size of the matrix.

2.1 Vargas's

This calculation method handles the multiple unit eigenvalue situation. (The easiest way to get into the multiple unit eigenvalue case, is to take a network with multiple sinks, and replace the sinks columns with identity columns). The formula is based upon this formula.

$$\begin{aligned} n_1 &= \text{Number of unit eigenvalues} \\ \Delta^i(\lambda) &= \text{The } i^{th}\text{derivative of the characteristic} \\ &\quad \text{polynomial, evaluated at } \lambda. \end{aligned}$$

$$W^\infty = n_1 \sum_{k=0}^{n_1} (-1)^k \frac{n_1!}{(n_1-k)!} \left[\frac{\Delta^{(n_1-k)}(\lambda)}{\Delta^{n_1}(\lambda)} (\lambda I - W)^{-k-1} \right]_{\lambda=1}.$$

However, this form of the formula is not very conducive to computer calculations. Using the fact that a matrix satisfies its own characteristic polynomial, Luis Vargas came up with the following derivation from this formula.

$$\begin{aligned} n_1 &= \text{The number of unit eigenvectors} \\ b_i &= \text{The degree } i \text{ piece of the characteristic polynomial of } A \\ \alpha_k &= \sum_{h=0}^{k-n+1} \frac{(k-h)!}{(k-n_1+1-h)!} b_{n-h} \\ W^\infty &= \sum_{k=n_1-1}^{n-1} \alpha_k W^{n-k-1} \end{aligned}$$

This is the formula actually used in the software when we use Vargas's formula. In every instance thus tested, this gives the same result as any other case. [2]

[2]The calculations to find all eigenvalues is directly taken from numerical recipes in c.

2.2 Out of date

This algorithms come from old times in the code. They are here for kicks, but should not be considered either safe, or effective. However they are interesting, at the very least, for seeing what goes wrong using these techniques. Here they are.

2.2.1 Pre-2000 Version

This code is part of the original limit matrix calculation code. In my defense, I wasn't paying a lot of attention to unusual cases at this time. Nor was I too worried about unbalanced hierarchies. This case does not handle unbalanced hierarchies, or sinks cases correctly. Other forms it should work correctly with. This code is not the same form that existed at all times before 2000, however it has many some similarities with that code. It is the closest remnant of that code left in the system. The algorithm goes something like this.

1. Calculate the 512^{th} power of the matrix. Call this *startPower*. We use *powerScale* to do this. This rescales after each power multiplication. If you have no sinks, this causes no problems.
2. If that power vanishes, consider it a hierarchy.
   (a) If we are a hierarchy, start back with the initial super matrix.
   (b) Keep taking large powers, until the matrix vanishes (this will happen at the number levels in the hierarchy power).
   (c) Return the power just before it vanishes as the limit matrix. This only works for balanced hierarchies.
3. If that power didn't vanish, start looking for convergence. Set *nextPower* to *startPower*.
   (a) Multiply *nextPower*, by the super matrix, and set *nextPower* to this new next power. Then normalize the result matrix, and use that as *nextPower*.
   (b) Compare (without scaling) *nextPower* to *startPower*. If they are close enough we have found a cycle [3]. Average each power in the cycle and return this as the limit matrix (notice the may only be one matrix to average, i.e. cycle length one).
   (c) If not cycling go back two steps, and start again. Do this n^2 times. If this never converges to a cycle, we simply return the zero matrix, and give you a warning.

[3] This cycle may be of length one, i.e. we have already converged at the 512^{th} power. The assumption is that the 512^{th} is large enough to get it to converge (which may not happen).

2.2.2 Pre-2001 Version

This name is a bit misleading, in that it was removed from the main code in early 2000, but since Pre-2000 was already taken, I used this name. This code is very similar to the Pre-2000 code, except scaling is done to check for convergence, but those scaled values are not stored. Here is the algorithm.

1. Compute a *startPower*, except do not normalize. The power we start with is given by the starting power algorithm "see 2.4.3 on page 9".
2. If *startPower* is zero, then we have a hierarchy.
   (a) If we are a hierarchy, start back with the initial super matrix.
   (b) Keep taking large powers, until the matrix vanishes (this will happen at the number levels in the hierarchy power).
   (c) Return the power just before it vanishes as the limit matrix. This only works for balanced hierarchies.
3. If *startPower* is not zero, we have a network. Set $nextPower = startPower \times W$. If *nextPower* equals *startPower*, we have converged, and we just return *nextPower* as the limit matrix. If not, we go into the following:
   (a) $nextPower = nextPower \times W$.
   (b) Scale *nextPower* and W temporarily, and compare these two.
      i. If they are not close return to previous step, using unscaled versions of these matrices for next matrix multiplication.
      ii. If they are close, we have found a cycle. Normalize each of the intermediate powers, and average them together to give the resulting limit matrix. Notice that is only at the end that we permanently normalize these results.
      iii. If we have done this more than n^2 times, there is no cycle, and we give up. We return the zero matrix, and report an error.

2.3 Current

These are methods that handle all hierarchies, and networks without sinks correctly. They all disagree on networks with sinks, but that's where the fun is anyway.

2.3.1 Calculus type

This version is very similar to the last. The only difference is it adds identity columns only if we had a hierarchy. Here is it layed out step by step.

1. If we don't have a hierarchy, change nothing [4].

[4] We figure out if we have a hierarchy by computing W^n, and seeing if it disappears. If it does we have a hierarchy. If not, we do not.

2. If we have a hierarchy, replace all sink columns by identity columns.

3. Call the function *SM_limitNonHierarchy*, "see 2.4.1 on page 8".

2.3.2 Scaling By Scalar

This is a bizarre calculation that I came up with. The idea behind it is that rather than scaling each column by a, perhaps different, number; I would scale the whole matrix by a scalar (which is the largest of those scaling numbers). This way it wouldn't matter whether I did the scaling at the beginning, or at the end of the calculations (since scalar multiplication is commutative). I wouldn't trust this one at all, but it is kind of fun to watch it. Here is the algorithm for multiplying matrices $A \times B$ used (this also applies to raising matrices to powers):

1. First calculate $A \times B$.

2. Find the maximum column sum of $A \times B$, and call this max.

3. Return $\frac{1}{max} A \times B$.

With that, here is the actual algorithm.

1. If this is a hierarchy, replace any zero columns in W identity columns (i.e. columns in an $n \times n$ matrix in the same place as the column we are looking at). We check for a hierarchy by taking the W^{n+1}, and see if it is zero. If so we have a hierarchy. If not we do not.

2. Start power sp is given by the starting power algorithm "see 2.4.3 on page 9".

3. Compute $startPower = W^{sp}$ (scaling by scalar), and set $nextPower = startPower$.

4. Loop over the following steps up to $100n$ times.

   (a) Compute $nextPower = nextPower \times W$.

   (b) If this is the same as $startPower$, we have found a cycle. We average all the intermediate ones, and return that as the limit matrix.

   (c) If not, and we have done this less than n times, go back two steps, and start again.

   (d) If not, and we have done this n times, break out of this loop.

5. Set $startPower$ to the last power calculated, and go back into the previous loop. Do this at most $100n$ times.

6. If we make it here, we never converged. Warn us that something went wrong, and return the zero matrix.

2.3.3 New Hierarchy

These two functions are the latest variants of our limit matrix calculations. They came into existence because we wanted to have intermediate synthesis information in the limit matrix for hierarchies. The standard tricks to synthesize using the limit matrix do not give this information, so a new algorithm had to be constructed. The new algorithm is:

1. Separate the network into hierarchy and non-hierarchy parts. This is done in the following way.
   (a) Any zero columns in the matrix come from the hierarchy part. If there are no zero columns we are done.
   (b) Remove the columns (and their corresponding rows) from the matrix (i.e. remove those nodes from the network).
   (c) Feed the resulting matrix back into step one.
2. Once we know the hierarchy and non-hierarchy part, reorganize the super matrix so that the last nodes are the ones from the hierarchy. The super matrix looks like:
$$W = \begin{pmatrix} A & 0 \\ C & B \end{pmatrix}$$
3. Renormalize the A and B matrices.
4. We compute A^∞ using *SM_limitNonHierarchy*, "see 2.4.1 on page 8".
5. We compute B^∞ using *SM_limitHierarchy*, "see 2.4.2 on page 8".
6. Here is where the two types of hierarchy calculations differ, in how they compute the lower corner, LC, of the resulting limit matrix.

   Without limit We compute the lower corner via the formula:
$$LC = C \times A^\infty + B^\infty \times C$$
   .

   With limit We compute the lower corner recursively. The formula is:
$$LC_0 = C$$
$$LC_{i+1} = LC_i \times A^\infty + B^\infty \times LC_i$$
   We wait for this to converge (if after n^2 iterations, we don't converge, we give up and return LC_{n^2}). These matrices will get smaller and smaller, in general, so we use a scaled comparison to check for convergence (but we don't store these scaled values). To get the final lower corner, we do renormalize (but only at the last possible step).
7. Finally this information is put together in the following matrix:
$$W^\infty = \begin{pmatrix} A^\infty & 0 \\ LC & B^\infty \end{pmatrix}$$
   We then renormalize this matrix, just like any other matrix.

2.3.4 Identity At Sinks

This version of the calculation is straight forward.

1. Find all sink columns (zero columns), and replace them by identity columns.
2. Call the function *SM_limitNonHierarchy*, "see 2.4.1 on page 8".

2.3.5 Sinks formula

The sinks formula was the first approximation I came up with to handle sinks differently than everyone else. There are two versions of this algorithm (straight normalize) and (normalize limitB). These two use the same calculations, except the normalize the limit matrix slightly differently. This algorithm does the following.

1. Find all sinks in the network (all columns which are zero).
2. Reorganize the nodes so that these nodes are the last columns of the of the matrix.
3. The matrix then has the form:

$$W = \begin{pmatrix} B & 0 \\ A & 0 \end{pmatrix}$$

   Where B is the matrix for the part of the network without the sinks, and A is how the network part connects to the sinks.

4. Normalize B.
5. We compute B^∞ using the *SM_limitPowerNG* function [5].
6. Then the limit matrix (before scaling) is given by:

$$W^{\infty\prime} = \begin{pmatrix} B^\infty & 0 \\ A \times B^\infty & 0 \end{pmatrix}$$

7. Then we scale, depending on the type of scaling chosen.

   Straight Normalize This just normalizes $W^{\infty\prime}$ by each column by the inverse of the column sum. This is the way I would normalize that matrix if I had no other information on it.

   Normalize limitB This normalizes the B^∞ part of the limit matrix (i.e. upper right hand corner). Since each column sum of B and A, when added together, add to one, I use the same logic to make the column sum of B^∞ and A, when added together, add to on.

[5] In other words the *Calculus type* algorithm.

2.4 Common Algorithms

Many of these algorithms contain common algorithms in them (for instance the *Calculus type*, and *Identity at sinks*). Here I would like to describe these.

2.4.1 Limit Non-Hierarchy

This algorithm handles non-hierarchies correctly. It will also handle sinks correctly (without putting identities in their place). The only thing this has problems is with hierarchies (i.e. things whose matrices W vanish after the n^{th} power). Here is the algorithm:

1. The starting power sp is given by the starting power algorithm "see 2.4.3 on page 9".
2. We compute $startPower = W^{sp}$.
3. We then enter the following loop (set $nextPower = startPower$)
   (a) Compute the nextPower by doing $nextPower = startPower \times W$.
   (b) Temporarily normalize $nextPower$ and $startPower$, and compare them (these scaled values are then immediately forgotten).
   (c) If $nextPower$ is the same as $startPower$, we have found a cycle (perhaps of length one). Now normalize each matrix in the cycle, and average them. Return this as the limit matrix.
   (d) If $nextPower$ is different from $startPower$, go back to the beginning of this loop. If, however, we have done this n times, break out of this loop and go on to the next step.
4. If we made it here, we haven't found a cycle, so go back to the beginning of the previous loop, with $nextPower =$ the last power you computed.
5. If you go through the previous loop $100n$ times, give up. Give a warning, and return the zero matrix.

2.4.2 Limit Hierarchy

This algorithm handles hierarchies correctly, and reports all intermediate synthesis values as well. The algorithm is simple, and will only be applied to hierarchies [6]

1. Compute all powers of W, until it vanishes (if we have a hierarchy, the maximum this could be is n). If it doesn't vanish at n, then you didn't have a hierarchy, but it will stop there anyway.
2. Add up all of the powers from the previous step.
3. Normalize the resulting sum, and return this as the limit matrix.

[6] If it is applied to a non-hierarchy, it will just average $W^1, W^2, ..., W^n$.

2.4.3 Starting Power

This is a cute little algorithm used in some limit matrix calculations to figure what power to start looking for convergence at. It may change in the future, to be more intelligent, however currently it does the following steps.

1. Let sp be the starting power we are looking for, for the matrix W.
2. First let $sp = 10 \times n^2$.
3. Choose the next largest power of two, and let that be sp (we use largest powers of two because matrix multiplication is more efficient that way).
4. If $sp > 1024$, let $sp = 1024$ (I don't want starting powers getting out of hand, altough this cut off may change later).
5. Check to see if W^{sp} is zero as far as the computer is concerned (due to round off).
6. If it was not zero, we have our starting power, and we stop here.
7. If it was zero, we then let $sp = 15 \times n$.
8. If $sp > 256$, we let $sp = 256$, and if $sp < 32$, we let $sp = 32$.
9. So if we made it here, $32 \leq sp \leq 256$.

3 Pros and Cons

Some of these algorithms excel in certain cases, and do terribly in other cases. Some do fair at all of them, and still others do poorly in most instances. I'd like to outline the strengths and weaknesses of each algorithm. Beside each I will give an overall score on a scale of one to ten.

Calculus Type (9.0) This algorithm is overall the best available. It handles all cases, and gives meaningful answers.

Strengths Handles networks without sinks correctly. Also handles networks with sinks in a fairly meaningful way (which never gives bizarre answers, like an alternative getting 100% for no reason). It does handle all hierarchies correctly. However it will not give intermediate hierarchy synthesized values.

Weaknesses Does not give intermediate hierarchy synthesized values.

Scaling By Scalar (8.5) This algorithm was designed on the spur of the moment. I had the idea to scale by scalars, instead of but matrix multiplication. That way it would be commutative. It appears to always work the same as the calculus type, and for good reason. With a little work the proof is straight-forward.

Strengths Same as *Calculus Type*. In addition it is rescaled always, which makes some folks more comfortable.

Weaknesses Same as *Calculus Type*.

New Hierarchy (8.0) This algorithm exists to give intermediate results of synthesis in the hierarchy case.

Strengths This algorithm gives the intermediate results for hierarchies. It also handles networks without sinks correctly always. Also handles degenerate hierarchy cases better than all other algorithms.

Weaknesses Its handling of networks with sinks is not at all canonical. The algorithm does stand on its own merits, but the results don't always work with the logic I would like to see.

Identity at Sinks (7) This algorithm works the same as the calculus type in the hierarchy case, and the network without sinks case. However if you have a network with sinks, it puts self-loops on the sinks.

Strengths Works well in hierarchy, and network without sinks case.

Weaknesses Does not give intermediate synthesized values back in the hierarchy case. Also in the network with sinks case, it places undo emphasis on the alternatives. This can result in an alternative getting 100% in cases where it makes no sense (see the nuclear missile defense model in our samples).

Sinks formula (6.5) This algorithm was the attempt just before the *New Hierarchy* algorithm. It never gives completely bizarre results (as a matter of a fact it tends to be very close to the *Calculus Type* algorithm). The reason it gets such a low value is that it fails to handle unbalanced hierarchies, because of weaknesses in the method of putting identity columns at sinks for hierarchies.

Strengths Is an easy to understand algorithm. It seems to separate the world into easy parts. It works fine with networks with sinks.

Weaknesses It does not handle unbalanced hierarchies correctly because of the logic of adding ones to the diagonal of sinks [7] Another weakness is in the logic of the algorithm. The *New Hierarchy* algorithm was developed to repair this. However this weakness turns out to be strength as well (since the simplifying assumption which bothers me, makes the overall calculation easier). Oh yes, it also doesn't give the intermediate synthesis values for a hierarchy.

[7] For the B^{∞} part, it uses the *Calculus Type* algorithm, which adds ones to diagonals of hierarchies. This causes our problems in unbalance hierarchy cases. However, if we used the function *SM_limitNonHierarchy* instead, this weakness would leave. Since this is mostly for experimental purposes, this change will not be made anytime soon.

Pre-2001 Version (3) This algorithm is a fairly stable algorithm for handling networks. It is not as sophisticated as the earlier mentioned ones. However it does has some extreme shortcomings, which is why it is rated so low.

Strengths Handles networks, and pure hierarchies correctly (but not unbalanced).

Weaknesses If the network has a hard time converging, this will bail before the other algorithms mentioned. It does not handle unbalanced hierarchies correctly. It also doesn't give the intermediate results for a hierarchy of any kind.

Pre-2000 Version (2) This algorithm is all that is left of the first version of the limit matrix calculation I first included in the code. It is juvenile, but does work for networks without sinks, which was all I thought about at that time.

Strengths Not many! It does handle networks without sinks correctly.

Weaknesses Every other case it will mess up. Never, ever trust this one!

4 Examples

To further explain these calculations, let us look at a few examples. These will illustrate where various algorithms have their weaknesses, or strengths.

4.1 Balanced Hierarchy

The following is the super matrix for simple balanced hierarchy. This sample is included in the sample models, with the name *simple_bal_hier.mod*.

$$\begin{pmatrix} & goal & cri2 & crit1 & alt1 & alt2 & alt3 \\ goal & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ cri2 & 0.250000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ crit1 & 0.750000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ alt1 & 0.000000 & 0.657069 & 0.124307 & 0.000000 & 0.000000 & 0.000000 \\ alt2 & 0.000000 & 0.196306 & 0.358558 & 0.000000 & 0.000000 & 0.000000 \\ alt3 & 0.000000 & 0.146626 & 0.517135 & 0.000000 & 0.000000 & 0.000000 \end{pmatrix}$$

The following are the limit matrices found from each algorithm, with a brief explanation of how they were found.

4.1.1 Calculus Type, and Identity at sinks

Both of the methods put self-loops on hierarchies, and then use the same algorithm from there. So they both give the following limit matrix.

	$goal$	$cri2$	$crit1$	$alt1$	$alt2$	$alt3$
$goal$	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
$cri2$	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
$crit1$	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
$alt1$	0.257497	0.657069	0.124307	1.000000	0.000000	0.000000
$alt2$	0.317995	0.196306	0.358558	0.000000	1.000000	0.000000
$alt3$	0.424508	0.146626	0.517135	0.000000	0.000000	1.000000

Basically, both of these algorithms replace the zero columns with identity columns, then take the limit matrix of that one. In this case they converge quickly, and everything works out well. The only thing missing is the priority for *cri1, cri2* with respect to *goal*.

4.1.2 Scaling By Scalar

This algorithm works the same as the *Calculus Type* in this case. The reason is that it puts self-loops on the sinks, and then computes the limit matrix, scaling using scalars. Since there all columns add to one here [8], there is no scaling to do. The exact results it gives are:

$W^{\infty} =$

	$goal$	$cri2$	$crit1$	$alt1$	$alt2$	$alt3$
$goal$	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
$cri2$	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
$crit1$	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
$alt1$	0.257497	0.657069	0.124307	1.000000	0.000000	0.000000
$alt2$	0.317995	0.196306	0.358558	0.000000	1.000000	0.000000
$alt3$	0.424508	0.146626	0.517135	0.000000	0.000000	1.000000

.

4.1.3 New Hierarchy

These two algorithms both give give the same answers for hierarchies always. Their limit matrices look like:

	$goal$	$cri2$	$crit1$	$alt1$	$alt2$	$alt3$
$goal$	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
$cri2$	0.125000	0.000000	0.000000	0.000000	0.000000	0.000000
$crit1$	0.375000	0.000000	0.000000	0.000000	0.000000	0.000000
$alt1$	0.128749	0.657069	0.124307	0.000000	0.000000	0.000000
$alt2$	0.158998	0.196306	0.358558	0.000000	0.000000	0.000000
$alt3$	0.212254	0.146626	0.517135	0.000000	0.000000	0.000000

[8] After we add identity columns on the sinks, of course.

If you check under the goal column, this gives the same alternative priorities as the previous methods. The only difference is we now have the information on the priority of criteria with respect to the goal node.

This result was found, by raising the super matrix to larger powers, until it vanishes. This one vanishes at the 3^{rd} power. Then I simply add those powers up, and rescale the resulting matrix. So I do:

$$W^{\infty} = Normalize(W + W^2).$$

4.1.4 Sinks formula

These both give the same result in this case. Every alternative is a sink in this case, so the A and B matrices from the algorithm are:

$$B = \begin{pmatrix} & goal & cri2 & crit1 \\ goal & 0.000000 & 0.000000 & 0.000000 \\ cri2 & 0.250000 & 0.000000 & 0.000000 \\ crit1 & 0.750000 & 0.000000 & 0.000000 \end{pmatrix}$$

$$A = \begin{pmatrix} & goal & cri2 & crit1 \\ alt1 & 0.000000 & 0.657069 & 0.124307 \\ alt2 & 0.000000 & 0.196306 & 0.358558 \\ alt3 & 0.000000 & 0.146626 & 0.517135 \end{pmatrix}$$

When we use the standard limit matrix routine (used in the *Calculus Type* routine), we get

$$B^{\infty} = \begin{pmatrix} & goal & cri2 & crit1 \\ goal & 0.000000 & 0.000000 & 0.000000 \\ cri2 & 0.250000 & 0.000000 & 0.000000 \\ crit1 & 0.750000 & 0.000000 & 0.000000 \end{pmatrix}$$

$$A \times B^{\infty} = \begin{pmatrix} & goal & cri2 & crit1 \\ alt1 & 0.257497 & 0.657069 & 0.124307 \\ alt2 & 0.317995 & 0.196306 & 0.358558 \\ alt3 & 0.424508 & 0.146626 & 0.517135 \end{pmatrix}$$

$$W^{\infty} = \begin{pmatrix} & goal & cri2 & crit1 & alt1 & alt2 & alt3 \\ goal & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ cri2 & 0.125000 & 0.500000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ crit1 & 0.375000 & 0.000000 & 0.500000 & 0.000000 & 0.000000 & 0.000000 \\ alt1 & 0.128749 & 0.328534 & 0.062153 & 0.000000 & 0.000000 & 0.000000 \\ alt2 & 0.158998 & 0.098153 & 0.179279 & 0.000000 & 0.000000 & 0.000000 \\ alt3 & 0.212254 & 0.073313 & 0.258568 & 0.000000 & 0.000000 & 0.000000 \end{pmatrix}$$

4.1.5 Pre2001, and Pre2000 Version

Both of these algorithms work the same on hierarchies. They get lucky here because this is a balanced hierarchy. They both take to power just before the

matrix vanishes, and return this as the limit. In this case this is the 2^{nd} power. The result they give is:

$$W^{\infty} = \begin{pmatrix} & goal & cri2 & crit1 & alt1 & alt2 & alt3 \\ goal & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ cri2 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ crit1 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ alt1 & 0.257497 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ alt2 & 0.317995 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ alt3 & 0.424508 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \end{pmatrix}$$

4.2 Unbalanced Hierarchy

The next example is an unbalanced hierarchy (one where all paths from alternatives to goal are not the same length). It is basically the previous model with some extra connections from the goal node directly to the alternatives. It can be found in the file *simple_unbal_hier.mod*.The scaled super matrix follows:

$$W = \begin{pmatrix} & goal & cri2 & crit1 & alt1 & alt2 & alt3 \\ goal & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ cri2 & 0.125000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ crit1 & 0.375000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ alt1 & 0.360768 & 0.657069 & 0.124307 & 0.000000 & 0.000000 & 0.000000 \\ alt2 & 0.092153 & 0.196306 & 0.358558 & 0.000000 & 0.000000 & 0.000000 \\ alt3 & 0.047078 & 0.146626 & 0.517135 & 0.000000 & 0.000000 & 0.000000 \end{pmatrix}$$

.

Since this case is very similar to the last one, I will not give as much explanation.

4.2.1 Calculus Type, and Identity at sinks

Both of the methods put self-loops on hierarchies, and then use the same algorithm. The result they give is the following.

$$W^{\infty} = \begin{pmatrix} & goal & cri2 & crit1 & alt1 & alt2 & alt3 \\ goal & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ cri2 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ crit1 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ alt1 & 0.489517 & 0.657069 & 0.124307 & 1.000000 & 0.000000 & 0.000000 \\ alt2 & 0.251151 & 0.196306 & 0.358558 & 0.000000 & 1.000000 & 0.000000 \\ alt3 & 0.259332 & 0.146626 & 0.517135 & 0.000000 & 0.000000 & 1.000000 \end{pmatrix}$$

$$Priorities = \begin{pmatrix} alt1 & alt2 & alt3 \\ 0.48952 & 0.25115 & 0.25933 \end{pmatrix}$$

4.2.2 Scaling By Scalar

Works fine in this case. Since it puts self loops on the alternatives, there is no scaling to worry about. Its results are:

$$W^{\infty} = \begin{pmatrix} & goal & cri2 & crit1 & alt1 & alt2 & alt3 \\ goal & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ cri2 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ crit1 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ alt1 & 0.489517 & 0.657069 & 0.124307 & 1.000000 & 0.000000 & 0.000000 \\ alt2 & 0.251151 & 0.196306 & 0.358558 & 0.000000 & 1.000000 & 0.000000 \\ alt3 & 0.259332 & 0.146626 & 0.517135 & 0.000000 & 0.000000 & 1.000000 \end{pmatrix}$$

The priorities given are

$$\begin{pmatrix} alt1 & alt2 & alt3 \\ 0.48952 & 0.25115 & 0.25933 \end{pmatrix}$$

4.2.3 New Hierarchy

As always with a hierarchy, these give all intermediate synthesis values as well. Both versions give the same result in this case. The matrix is (this was arrived at by doing $Normalize(W + W^2)$, since W^3 is the first power that vanishes):

$$W^{\infty} = \begin{pmatrix} & goal & cri2 & crit1 & alt1 & alt2 & alt3 \\ goal & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ cri2 & 0.083333 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ crit1 & 0.250000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ alt1 & 0.326344 & 0.657069 & 0.124307 & 0.000000 & 0.000000 & 0.000000 \\ alt2 & 0.167434 & 0.196306 & 0.358558 & 0.000000 & 0.000000 & 0.000000 \\ alt3 & 0.172888 & 0.146626 & 0.517135 & 0.000000 & 0.000000 & 0.000000 \end{pmatrix}$$

The priorities are

$$\begin{pmatrix} alt1 & alt2 & alt3 \\ 0.48952 & 0.25115 & 0.25933 \end{pmatrix}$$

4.2.4 Sinks Formula

This formula fails here because of the unbalanced nature of the system. The reason can be seen by looking at the following matrices.

$$B^{\infty} = \begin{pmatrix} 0.000000 & 0.000000 & 0.000000 \\ 0.250000 & 1.000000 & 0.000000 \\ 0.750000 & 0.000000 & 1.000000 \end{pmatrix}$$

$$A = \begin{pmatrix} 0.360768 & 0.657069 & 0.124307 \\ 0.092153 & 0.196306 & 0.358558 \\ 0.047078 & 0.146626 & 0.517135 \end{pmatrix}$$

Once we multiply these to give the lower corner, and normalize, we get:

$$W^{\infty} = \begin{pmatrix} & goal & cri2 & crit1 & alt1 & alt2 & alt3 \\ goal & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ cri2 & 0.125000 & 0.500000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ crit1 & 0.375000 & 0.000000 & 0.500000 & 0.000000 & 0.000000 & 0.000000 \\ alt1 & 0.128749 & 0.328534 & 0.062153 & 0.000000 & 0.000000 & 0.000000 \\ alt2 & 0.158998 & 0.098153 & 0.179279 & 0.000000 & 0.000000 & 0.000000 \\ alt3 & 0.212254 & 0.073313 & 0.258568 & 0.000000 & 0.000000 & 0.000000 \end{pmatrix}$$

The priorities are

$$\begin{pmatrix} alt1 & alt2 & alt3 \\ 0.25750 & 0.31800 & 0.42451 \end{pmatrix}$$

Which essentially forgets about the intermediate values. This is caused directly by a flaw in this system, because the goal directly has priorities. Another flaw, which would crop up in other unbalanced situations has to do with the flaw in putting 1's on the diagonal of sinks to do limit matrix calculations (for doing the B^{∞} calculation). Please see the file *simple_unbal_hier2.mod* for an example.

4.2.5 Pre-2000 and Pre-2001 versions

Both run into the same problem as the *Sinks formula* in this case, although they get at their answers in completely different ways. These simple notice the the third power of the super matrix is where everything vanishes, so they return the second power as the limit matrix. The matrix, and priorities they return are:

$$W^{\infty} = \begin{pmatrix} & goal & cri2 & crit1 & alt1 & alt2 & alt3 \\ goal & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ cri2 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ crit1 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ alt1 & 0.257497 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ alt2 & 0.317995 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ alt3 & 0.424508 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \end{pmatrix}$$

The priorities which result are

$$\begin{pmatrix} alt1 & alt2 & alt3 \\ 0.25750 & 0.31800 & 0.42451 \end{pmatrix}$$

4.3 Odd hierarchy

This is the first hierarchy given, except I add in a dummy alternative that connects to nothing and from nothing. It is simply dead. The file is *simple_bal_hier2.mod*. There are two valid ways to think of the result you should get.

- You should get the zero out for its priority, since he isn't connected to anything. He has no effect on anything, so he really ought to be zero.
- You should get 50% for him, since you know nothing about him, he gets half, and the rest get half.

It turns out that the *New Hierarchy*, *Sinks formula*, and the *Pre-2000, Pre-2001 versions* all give former result. Whereas the rest give the latter. This tends to make be believe in the *New Hierarchy* method more for hierarchies. I know this is a degenerate case, but it is of some interest though.

4.4 Network with sinks 1

This is a trivial network with sinks. It has no really complicated structure, and will give us a false sense of security, beware. Nonetheless it does differentiate between some of the versions of these algorithms (in particular the *Identity at sinks* and *Calculus Type*). It can be found in the file *net_sinks1.mod*, and here is its scaled super matrix.

$$\begin{pmatrix} & co1-1 & co1-2 & co2-1 & co2-2 & a1 & a2 \\ co1-1 & 0.000000 & 0.000000 & 0.218750 & 0.218750 & 0.000000 & 0.000000 \\ co1-2 & 0.000000 & 0.000000 & 0.031250 & 0.031250 & 0.000000 & 0.000000 \\ co2-1 & 0.666667 & 0.625000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ co2-2 & 0.166667 & 0.208333 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ a1 & 0.055556 & 0.133333 & 0.250000 & 0.675000 & 0.000000 & 0.000000 \\ a2 & 0.111111 & 0.033333 & 0.500000 & 0.075000 & 0.000000 & 0.000000 \end{pmatrix}$$

4.4.1 Calculus Type

This algorithm will not put self-loops in. Instead it does a calculus type calculation. The result limit matrix, and priority vector are:

$$W^{\infty} = \begin{pmatrix} & co1-1 & co1-2 & co2-1 & co2-2 & a1 & a2 \\ co1-1 & 0.109375 & 0.109375 & 0.109375 & 0.109375 & 0.000000 & 0.000000 \\ co1-2 & 0.015625 & 0.015625 & 0.015625 & 0.015625 & 0.000000 & 0.000000 \\ co2-1 & 0.330729 & 0.330729 & 0.330729 & 0.330729 & 0.000000 & 0.000000 \\ co2-2 & 0.085938 & 0.085938 & 0.085938 & 0.085938 & 0.000000 & 0.000000 \\ a1 & 0.201467 & 0.201467 & 0.201467 & 0.201467 & 0.000000 & 0.000000 \\ a2 & 0.256866 & 0.256866 & 0.256866 & 0.256866 & 0.000000 & 0.000000 \end{pmatrix}$$

$$\begin{pmatrix} a1 & a2 \\ 0.43956 & 0.56044 \end{pmatrix}$$

4.4.2 Scaling By Scalar

Again this gives exactly the same answer as the *Calculus Type*, including the limit matrix. I believe this algorithm works out identically to the *Calculus Type*, as these examples seem to indicate.

4.4.3 Identity At Sinks

This algorithm gives a fairly different answer. The reason is that those identities at the sinks puts more emphasis on their structure, and less on the rest of the network. The resulting values are:

$$W^{\infty} = \begin{pmatrix} & co1-1 & co1-2 & co2-1 & co2-2 & a1 & a2 \\ co1-1 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ co1-2 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ co2-1 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ co2-2 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ a1 & 0.425948 & 0.521434 & 0.359471 & 0.784471 & 1.000000 & 0.000000 \\ a2 & 0.574052 & 0.478566 & 0.640529 & 0.215529 & 0.000000 & 1.000000 \end{pmatrix}$$

$$\begin{pmatrix} a1 & a2 \\ 0.52883 & 0.47717 \end{pmatrix}$$

4.4.4 New Hierarchy

Both of these algorithms give the same answer. You will notice that the limit matrix is different for any of the previous two methods. However the priority derived is the same as the *Calculus Type*.

$$W^{\infty} = \begin{pmatrix} & co1-1 & co1-2 & co2-1 & co2-2 & a1 & a2 \\ co1-1 & 0.198864 & 0.198864 & 0.087500 & 0.087500 & 0.000000 & 0.000000 \\ co1-2 & 0.028409 & 0.028409 & 0.012500 & 0.012500 & 0.000000 & 0.000000 \\ co2-1 & 0.180398 & 0.180398 & 0.079375 & 0.079375 & 0.000000 & 0.000000 \\ co2-2 & 0.046875 & 0.046875 & 0.020625 & 0.020625 & 0.000000 & 0.000000 \\ a1 & 0.239762 & 0.239762 & 0.351652 & 0.351652 & 0.000000 & 0.000000 \\ a2 & 0.305692 & 0.305692 & 0.448348 & 0.448348 & 0.000000 & 0.000000 \end{pmatrix}$$

$$\begin{pmatrix} a1 & a2 \\ 0.43956 & 0.56044 \end{pmatrix}$$

4.4.5 Sinks formula

These two give the same priorities as the *Calculus Type* and the *New Hierarchy*, but the limit matrices look different. Here they are:

For the straight normalize version

$$W^{\infty} = \begin{pmatrix} & co1-1 & co1-2 & co2-1 & co2-2 & a1 & a2 \\ co1-1 & 0.300000 & 0.300000 & 0.300000 & 0.300000 & 0.000000 & 0.000000 \\ co1-2 & 0.042857 & 0.042857 & 0.042857 & 0.042857 & 0.000000 & 0.000000 \\ co2-1 & 0.272143 & 0.272143 & 0.272143 & 0.272143 & 0.000000 & 0.000000 \\ co2-2 & 0.070714 & 0.070714 & 0.070714 & 0.070714 & 0.000000 & 0.000000 \\ a1 & 0.138149 & 0.138149 & 0.138149 & 0.138149 & 0.000000 & 0.000000 \\ a2 & 0.176137 & 0.176137 & 0.176137 & 0.176137 & 0.000000 & 0.000000 \end{pmatrix}$$

$$\begin{pmatrix} a1 & a2 \\ 0.43956 & 0.56044 \end{pmatrix}$$

For the normalize limitB part we get

$$W^{\infty} = \begin{pmatrix} & co1-1 & co1-2 & co2-1 & co2-2 & a1 & a2 \\ co1-1 & 0.236979 & 0.236979 & 0.236979 & 0.236979 & 0.000000 & 0.000000 \\ co1-2 & 0.033854 & 0.033854 & 0.033854 & 0.033854 & 0.000000 & 0.000000 \\ co2-1 & 0.214974 & 0.214974 & 0.214974 & 0.214974 & 0.000000 & 0.000000 \\ co2-2 & 0.055859 & 0.055859 & 0.055859 & 0.055859 & 0.000000 & 0.000000 \\ a1 & 0.201467 & 0.201467 & 0.201467 & 0.201467 & 0.000000 & 0.000000 \\ a2 & 0.256866 & 0.256866 & 0.256866 & 0.256866 & 0.000000 & 0.000000 \end{pmatrix}$$

$$\begin{pmatrix} a1 & a2 \\ 0.43956 & 0.56044 \end{pmatrix}$$

4.4.6 Others

The last two, *Pre-2001 Version, Pre-2000 Version* all give the same results as the *Calculus Type* in this case (both for priority, and actual limit matrix).

4.5 Network with sinks 2

This is the same network as the last one, except I will connect the alternatives back to the criteria *co2-2, co2-1*. Then I will add a sink alternative too. This one shows the first truly odd behavior of the identity at sinks model, where certain things get 100% even though they don't appear that they should. It also points out odd behavior in other algorithms as well. This network can be found in *net_sinks2.mod*, its super matrix is:

$$\begin{pmatrix} & co1-1 & co1-2 & co2-1 & co2-2 & a1 & a2 & sink \\ co1-1 & 0.000000 & 0.000000 & 0.218750 & 0.218750 & 0.000000 & 0.000000 & 0.000000 \\ co1-2 & 0.000000 & 0.000000 & 0.031250 & 0.031250 & 0.000000 & 0.000000 & 0.000000 \\ co2-1 & 0.666667 & 0.625000 & 0.000000 & 0.000000 & 0.800000 & 0.333333 & 0.000000 \\ co2-2 & 0.166667 & 0.208333 & 0.000000 & 0.000000 & 0.200000 & 0.666667 & 0.000000 \\ a1 & 0.055556 & 0.133333 & 0.250000 & 0.572034 & 0.000000 & 0.000000 & 0.000000 \\ a2 & 0.111111 & 0.033333 & 0.500000 & 0.063559 & 0.000000 & 0.000000 & 0.000000 \\ sink & 0.000000 & 0.000000 & 0.000000 & 0.114407 & 0.000000 & 0.000000 & 0.000000 \end{pmatrix}$$

4.5.1 Calculus Type

This method gives an interesting answer. The result is:

$$\begin{pmatrix} & co1-1 & co1-2 & co2-1 & co2-2 & a1 & a2 & sink \\ co1-1 & 0.108227 & 0.108227 & 0.108227 & 0.108227 & 0.108227 & 0.108227 & 0.000000 \\ co1-2 & 0.015461 & 0.015461 & 0.015461 & 0.015461 & 0.015461 & 0.015461 & 0.000000 \\ co2-1 & 0.301472 & 0.301472 & 0.301472 & 0.301472 & 0.301472 & 0.301472 & 0.000000 \\ co2-2 & 0.182712 & 0.182712 & 0.182712 & 0.182712 & 0.182712 & 0.182712 & 0.000000 \\ a1 & 0.192062 & 0.192062 & 0.192062 & 0.192062 & 0.192062 & 0.192062 & 0.000000 \\ a2 & 0.178707 & 0.178707 & 0.178707 & 0.178707 & 0.178707 & 0.178707 & 0.000000 \\ sink & 0.021360 & 0.021360 & 0.021360 & 0.021360 & 0.021360 & 0.021360 & 0.000000 \end{pmatrix}$$

$$\begin{pmatrix} a1 & a2 & sink \\ 0.48979 & 0.45574 & 0.05447 \end{pmatrix}$$

4.5.2 Scaling By Scalar

Once again this gives exactly the same answer as the *Calculus Type*, including the limit matrix.

4.5.3 Identity At Sinks

This algorithm makes the sink node into 100%, because of the artificial self-loop. Here is the limit matrix, and priority it comes up with.

$$\begin{pmatrix} & co1-1 & co1-2 & co2-1 & co2-2 & a1 & a2 & sink \\ co1-1 & 0.000015 & 0.000015 & 0.000015 & 0.000014 & 0.000015 & 0.000015 & 0.000000 \\ co1-2 & 0.000002 & 0.000002 & 0.000002 & 0.000002 & 0.000002 & 0.000002 & 0.000000 \\ co2-1 & 0.000042 & 0.000042 & 0.000042 & 0.000038 & 0.000042 & 0.000041 & 0.000000 \\ co2-2 & 0.000026 & 0.000026 & 0.000026 & 0.000023 & 0.000026 & 0.000025 & 0.000000 \\ a1 & 0.000027 & 0.000027 & 0.000027 & 0.000024 & 0.000027 & 0.000026 & 0.000000 \\ a2 & 0.000025 & 0.000025 & 0.000025 & 0.000023 & 0.000025 & 0.000024 & 0.000000 \\ sink & 0.999862 & 0.999862 & 0.999862 & 0.999876 & 0.999862 & 0.999868 & 1.000000 \end{pmatrix}$$

$$\begin{pmatrix} a1 & a2 & sink \\ 0 & 0 & 1 \end{pmatrix}$$

Notice that this is when the limit power error is set to 1.0×10^{06}, if you set it smaller, you will lose those trailing decimal places.

4.5.4 New Hierarchy

Both versions give the same result. It is an odd result, because the sink node gets most of the information, however there is still some information left in the

ratios of the other two. The result limit matrix, and priorities are:

$$\begin{pmatrix} & co1-1 & co1-2 & co2-1 & co2-2 & a1 & a2 & sink \\ co1-1 & 0.056095 & 0.056095 & 0.056095 & 0.052692 & 0.056095 & 0.056095 & 0.000000 \\ co1-2 & 0.008014 & 0.008014 & 0.008014 & 0.007527 & 0.008014 & 0.008014 & 0.000000 \\ co2-1 & 0.153491 & 0.153491 & 0.153491 & 0.144178 & 0.153491 & 0.153491 & 0.000000 \\ co2-2 & 0.091167 & 0.091167 & 0.091167 & 0.085635 & 0.091167 & 0.091167 & 0.000000 \\ a1 & 0.101445 & 0.101445 & 0.101445 & 0.095290 & 0.101445 & 0.101445 & 0.000000 \\ a2 & 0.089788 & 0.089788 & 0.089788 & 0.084341 & 0.089789 & 0.089789 & 0.000000 \\ sink & 0.500000 & 0.500000 & 0.500000 & 0.530337 & 0.500000 & 0.500000 & 0.000000 \end{pmatrix}$$

$$\begin{pmatrix} a1 & a2 & sink \\ 0.14462 & 0.12800 & 0.72737 \end{pmatrix}$$

4.5.5 Sinks formula (straight normalize)

This result is similar to the *Calculus Type* (where the sink gets little priority). However there are some differences.

$$\begin{pmatrix} & co1-1 & co1-2 & co2-1 & co2-2 & a1 & a2 & sink \\ co1-1 & 0.109898 & 0.109898 & 0.109898 & 0.109898 & 0.109898 & 0.109898 & 0.000000 \\ co1-2 & 0.015700 & 0.015700 & 0.015700 & 0.015700 & 0.015700 & 0.015700 & 0.000000 \\ co2-1 & 0.300709 & 0.300709 & 0.300709 & 0.300709 & 0.300709 & 0.300709 & 0.000000 \\ co2-2 & 0.178608 & 0.178608 & 0.178608 & 0.178608 & 0.178607 & 0.178607 & 0.000000 \\ a1 & 0.198745 & 0.198745 & 0.198744 & 0.198744 & 0.198745 & 0.198745 & 0.000000 \\ a2 & 0.175908 & 0.175908 & 0.175907 & 0.175907 & 0.175908 & 0.175908 & 0.000000 \\ sink & 0.020434 & 0.020434 & 0.020434 & 0.020434 & 0.020434 & 0.020434 & 0.000000 \end{pmatrix}$$

$$\begin{pmatrix} a1 & a2 & sink \\ 0.50304 & 0.44524 & 0.05171 \end{pmatrix}$$

4.5.6 Sinks formula (normalize limitB)

This is the first cases where the sinks formulae differ from each other (although only many decimal places out). Here is the output it gives.

$$\begin{pmatrix} & co1-1 & co1-2 & co2-1 & co2-2 & a1 & a2 & sink \\ co1-1 & 0.109850 & 0.109850 & 0.109850 & 0.109850 & 0.109850 & 0.109850 & 0.000000 \\ co1-2 & 0.015693 & 0.015693 & 0.015693 & 0.015693 & 0.015693 & 0.015693 & 0.000000 \\ co2-1 & 0.300578 & 0.300578 & 0.300578 & 0.300578 & 0.300578 & 0.300578 & 0.000000 \\ co2-2 & 0.178530 & 0.178530 & 0.178530 & 0.178530 & 0.178530 & 0.178530 & 0.000000 \\ a1 & 0.198658 & 0.198658 & 0.198658 & 0.198658 & 0.198658 & 0.198658 & 0.000000 \\ a2 & 0.175831 & 0.175831 & 0.175831 & 0.175831 & 0.175831 & 0.175831 & 0.000000 \\ sink & 0.020860 & 0.020860 & 0.020860 & 0.020860 & 0.020860 & 0.020860 & 0.000000 \end{pmatrix}$$

$$\begin{pmatrix} a1 & a2 & sink \\ 0.50249 & 0.44475 & 0.05726 \end{pmatrix}$$

4.5.7 Pre-2001 Version

This one gives the same as the *Calculus Type* now that I have changed the starting power algorithm. Before that point it would not converge.

4.5.8 Pre-2000 Version

Since this version always starts with the 512^{th} power, we are in good shape with it. It will converge, and gives different results than everyone else too. Here they are:

$$\begin{pmatrix} & co1-1 & co1-2 & co2-1 & co2-2 & a1 & a2 & sink \\ co1-1 & 0.000000 & 0.000000 & 0.218750 & 0.218750 & 0.000000 & 0.000000 & 0.000000 \\ co1-2 & 0.000000 & 0.000000 & 0.031250 & 0.031250 & 0.000000 & 0.000000 & 0.000000 \\ co2-1 & 0.666667 & 0.625000 & 0.000000 & 0.000000 & 0.800000 & 0.333333 & 0.000000 \\ co2-2 & 0.166667 & 0.208333 & 0.000000 & 0.000000 & 0.200000 & 0.666667 & 0.000000 \\ a1 & 0.055556 & 0.133333 & 0.250000 & 0.572034 & 0.000000 & 0.000000 & 0.000000 \\ a2 & 0.111111 & 0.033333 & 0.500000 & 0.063559 & 0.000000 & 0.000000 & 0.000000 \\ sink & 0.000000 & 0.000000 & 0.000000 & 0.114407 & 0.000000 & 0.000000 & 0.000000 \end{pmatrix}$$

$$\begin{pmatrix} a1 & a2 & sink \\ 0.48873 & 0.45732 & 0.05395 \end{pmatrix}$$

5 Priority Calculations

Just thought I'd mention how priority vectors are found from the limit matrix. It is simple. I average all goal columns together. A goal column, is the column of a node which has connections from it, but none going to it (except possibly a self connection).

If there are no goal columns, then I average all non-zero, non-identity columns (an identity column is a zero column, except it has a 1 on the diagonal).