Reconstruction of the Dirichlet to Neumann Map for a Resistor Network

Adrian V. Mariano
University of Washington

Miriam A. Myjak
Seattle University

June 21, 2003

Abstract

We describe methods of reconstructing partial Dirichlet to Neumann maps for square resistor networks. We consider possible uses and potential pitfalls inherent in the algorithm.

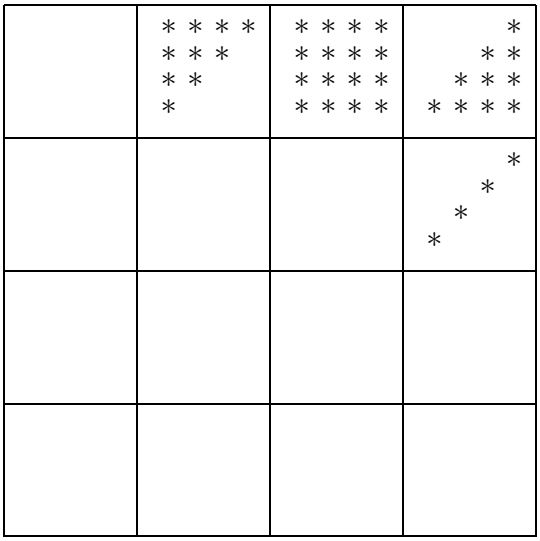
1 Introduction

We consider square resistor networks in the plane. In the forward Dirichlet to Neumann problem, the values of currents flowing through boundary nodes of an order n network are calculated given voltages on the boundary nodes and the conductivities γ of the resistors in the network. ¿From the solution of this problem, a matrix Λ is generated which maps voltages to currents at the boundary nodes. Curtis and Morrow showed that it is possible to parameterize Λ using a set of $2n(n+1)$ values from Λ — call this the standard set — and that Λ can be reconstructed from the standard set using various algebraic relations that hold for sets of elements in Λ [1]. We wrote FORTRAN computer programs to reconstruct the entire Λ from an incomplete Λ. Our first program reconstructed based on the standard parameter set using the method described in [1]. A later program used the algebraic relations of Λ to reconstruct as much the of the Λ as possible from any initial parameter set.

Using our reconstruction programs, we examined the accuracy of the resultant Λ's. We ran test cases to determine the influence of errors in the parameter set, and attempted to develop a method of increasing the accuracy of γ's obtained from inaccurate Λ's.

2 The Standard Set

The first program that we wrote, DRECON.F took the standard parameter set and reconstructed the whole Λ from it. The positions of the parameters are indicated by the *'s in the following figure (illustrated with $n = 4$).



2.1 Round Off Error

In order to determine the limitations of the reconstruction process, we used parameters from an accurate Λ for all $\gamma = 1$ to obtain a reconstructed matrix $\tilde{\Lambda}$. We attempted to reconstruct Λ for large n from accurate data using 14 digits of precision. To evaluate accuracy, we examined the following quantities:

$$E_{\lambda_{i,j}} = \left|\lambda_{i,j} - \tilde{\lambda}_{i,j}\right|$$

$$\overline{E_\lambda} = \frac{\sum\limits_{i \neq j} E_{\lambda_{i,j}}}{16n^2 - 4n}$$

$$\sigma_\lambda = \sqrt{\sum_{i \neq j} E^2_{\lambda_{i,j}}}$$

Here is sample error data from attempts to reconstruct an order n Λ for all γ's$= 1$

n	$E_{\lambda_{max}}$	$\overline{E_\lambda}$	σ_λ
10	0.0003912898	2.077439×10^{-6}	0.0008279846
11	0.007238837	3.179374×10^{-5}	0.01527787
12	16.76129	0.08454078	37.08824

When $\tilde{\Lambda}$ was used to recover γ for all Ω_1 using inverse code we wrote (the inverse algorithm is described in [1]), we obtained reasonable results for $n = 10$, but poor values for $n = 11$ and meaningless values for $n = 12$. ¿From these cases, we conclude that, unless higher precision calculations are available, round off error makes the reconstruction process useless for $n > 10$. As above, we evaluated accuracy by examining the following:

$$E_{\gamma_i} = |\gamma_i - \tilde{\gamma}_i|$$

$$\overline{E_\gamma} = \frac{\sum\limits_{i \in \Omega_1} E_{\gamma_i}}{2n(n+1)}$$

$$\sigma_\gamma = \sqrt{\sum_{i \in \Omega_1} E_{\gamma_i}^2}$$

When γ was obtained directly from Λ, we obtained

n	$E_{\gamma_{max}}$	$\overline{E_\gamma}$	σ_γ
10	0.0010833	0.00010139	0.0038351
11	0.033531	0.002264	0.10202
12	0.9197	0.047134	2.5738

When γ was obtained from $\tilde{\Lambda}$ we got

n	$E_{\gamma_{max}}$	$\overline{E_\gamma}$	σ_γ
10	0.010489	0.00044368	0.021024
11	19.452	0.20993	26.703
12	4.7978×10^{80}	3.0755×10^{78}	6.7851×10^{80}

2.2 Introduced Errors

We considered the impact of adding errors to Λ. For several different cases with $n = 3$ we added an error of .1 to one of the elements in the parameter set. The following figures display the top section of the standard parameter set. In the figures, the 24 λ's in the standard set were divided into groups of six according to σ_λ resulting from an error at that location. the six λ's with the lowest σ_λ are indicated by "$\cdot$". The

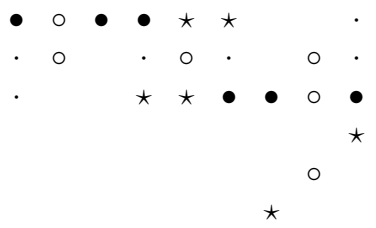


Figure 1: Constant conductors

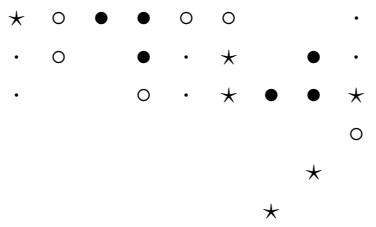


Figure 2: Conductors vary from .3 to 21

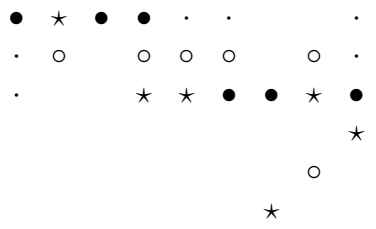


Figure 3: Conductors vary from 1 to 49

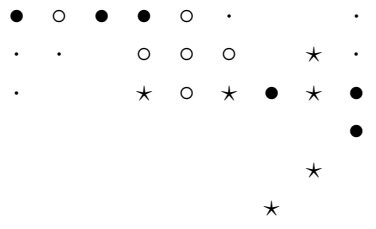


Figure 4: Conductors vary from 1 to 9

next group is indicated by "$\circ$", and the next by "$\star$". The six elements which yielded the largest error are labeled as "$\bullet$".

Changes at elements (1,4), (1,6), (1,7), (3,9), (3,10), and (3,12) consistently yielded large errors in Λ. These values are used immediately in the reconstruction to form ratios which we believe explains the sensitivity of the algorithm to variations there. Changes at (2,4), (3,4), (1,12), and (2,12) had a minimal effect on the reconstruction, reflecting the infrequent use that these elements get during the reconstruction.

3 Generalized Reconstruction

Kirchoff's Law states that the sum of currents entering a node is zero. The algebraic relations from this law are summarized in [1] as

$$\lambda_{i,m} + \sum_{j=1}^{k} \lambda_{i,j}\alpha_j = 0 \tag{1}$$

with i, j, k, m integers, $1 \leq k \leq n$, $m = 4n - k + 1$, and $k < i < m$. Recall that $\lambda_{i,j}$ is the current at boundary node i due to a voltage of one at boundary node j with zero potential on all other boundary nodes; α_j is the potential at the jth boundary node. The sum of any row or column of Λ is thus zero because the net current entering the network must be zero.

The FORTRAN code LAMFILL.F uses equation (1). A complete code listing is given in Appendix A. Sample input and output files are contained in Appendix B. First, LAMFILL.F calls the input subroutine LAMIN.F which reads input from fort.10 and fort.11. The complete Λ, if known, is contained in fort.10. (The forward Dirichlet program, FORDIRB.F, generates a Λ in the correct format.) The order n of the network appears on the first line of fort.10 followed by the rows of the matrix with one column-element appearing per line and one blank line separating different rows of the matrix. The purpose of reading the entire Λ is to compare the original matrix to the reconstructed matrix. The fort.11 file has the same form as the fort.10 file except that all elements have been zeroed out except the chosen parameters from which the reconstruction proceeds. A fort.11 file can be created using the filter program which reads in a Λ and a template file. A template file consists of the order, n, of the

problem, followed by $4n$ lines of $4n$ characters. A "." in the template file excludes the corresponding λ from the output. Any other character includes the element as an initial parameter used for the reconstruction.

In LAMIN.F, and throughout the program, the symmetry property of Λ is used to fill in the symmetric element of any new entry that is found. Further, a running total is kept of how many entries of Λ are known. Equation (1) gives relationships around a single corner. Similar relationships exist for the other corners, and all of the relationships have a form going clockwise around the resistor network, with a corresponding counterclockwise form. Thus, for a given k, eight different sets of equations of the form of (1) are possible.

In order to reconstruct new elements, λ, the α_j in (1) must be known. LAMFILL.F calls ALPHSOL.F, a subroutine designed to solve for the α_j. For a given corner in a given direction (clockwise or counterclockwise), the code checks to see if all $\lambda_{i,j}$ and the $\lambda_{i,m}$ are nonzero for a given value of i. If this is the case, the equation is stored. After checking all i, if a set of k or more equations has been found, EQCHEQ.F is called. For $k > 1$ (only one equation is required for $k = 1$), EQCHEQ can generate all possible combinations of the set of equations taken k at a time. Each new combination is sent to DGEFS.F, a double-precision linear solving routine to solve for the α_j. If the system is singular (as can happen, for example, if two of the equations are ratios of each other), the subroutine sets IND $= -4$, or, if the answer has potentially low significance, then DGEFS.F sets IND $= -10$. This particular system is rejected and the next combination is tried.

If some α's have been solved for, the next subroutine called is PNTSSOL.F. The program starts at the level of equations which has $k = 1$ the "ratio" equations. If the required α's are known for a given corner in a given direction, PNTSSOL.F checks to see if all but one λ appearing in the relevant equation from (1) are known. If this is the case, the unknown element is solved for and put into the reconstructed Λ. The program tries all ratio ($k = 1$) equations. If new elements are found, the code returns to the ALPHSOL routine to see if new α's can be found on the $k = 1$ level and PNTSSOL is called again. When no new λ's are found by PNTSSOL, the code advances to $k = 2$. At this level, the same procedure of solving for α's then solving for new λ's is followed. If any new λ's are found, the program goes back to the lowest

$k = 1$ level and repeats the whole procedure. In this manner, the program makes use of the smallest possible linear systems (which appear in solving for the α's) to solve for a given element of reconstructed Λ.

When the code can not solve for any new λ entries, DIAGSOL.F is called. Here, the fact that rows should add up to zero is used to solve for diagonal entries. If all but the diagonal entry is known in a given row, it is solved for.

The output procedure is LAMOUT.F. The order of the network, number of elements in the initial input file, and how many were reconstructed are printed at the top of the fort.12 output file. Then, three columns of data follow. The first column contains whatever was in the fort.10 file, the second contains the reconstructed elements (zero indicates this element was not solved for), and the final column is the difference of the first two columns.

One other output file, fort.16, containing the order of the network followed by the reconstructed matrix is created for use in inverse and data analysis routines.

4 Other Parameter Sets

Clearly the standard parameter set is not unique. We tried 1620 different partial Λ's with $2n(n+1)$ elements for $n = 3$ and $n = 6$. To generate partial Λ's, we divided Λ into 16 $n \times n$ squares and permuted solid blocks, upper and lower triangles in the left and right, and a diagonal line through the six $n \times n$ blocks above the diagonal. We generated the parameter sets in eight runs with different types of blocks to permute. Each run generated 180 templates (except for run eight which generated 360). The first digit of a parameter set indicates the run number and the three remaining digits indicated which template from that run. For $n = 3$ we obtained 177 parameter sets (60 of these are listed in Appendix C), and for $n = 6$ we got 172. Six of the parameter sets for $n = 6$ did not have corresponding sets for $n = 3$. Eleven of the sets for $n = 3$ lacked corresponding sets for $n = 6$. Figures 5 and 6 show example of parameter sets with this feature, and Appendix B contains part of the reconstruction output for these, and the corresponding paramter sets which fail.

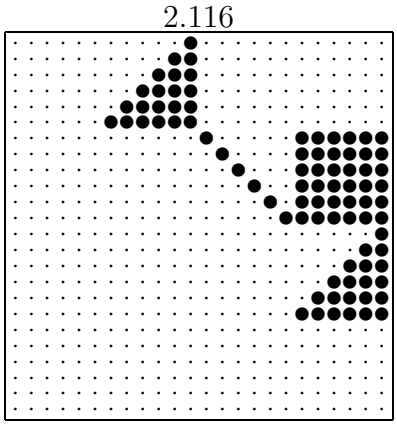


Figure 5: This pattern has no analog for $n = 3$

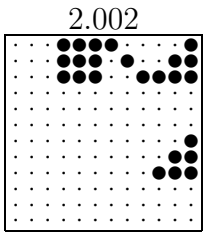


Figure 6: This pattern has no analog for $n = 5$ or $n = 6$

5 Restoring From Faulty Data

Given a Λ which contained errors, we attempted to reconstruct the γ's from it by using a minimum distance procedure. Using each of the available parameter sets, we obtained $\tilde{\Lambda}$ and calculated the distance between Λ and $\tilde{\Lambda}$. We used three different definitions of distance: $\overline{E_\lambda}$, $E_{\lambda_{max}}$, and σ_λ. From the $\tilde{\Lambda}$'s we obtained $\tilde{\gamma}$'s. We had hoped that $\tilde{\Lambda}$'s which were close to Λ would yield $\tilde{\gamma}$'s close to γ. The tables below are based on the ten closest $\tilde{\Lambda}$ and $\tilde{\gamma}$ for six different cases. The number on the left of each table is the index to the parameter set we used (the parameter sets are listed in Appendix C). The number at the bottom of each table is the γ obtained by directly from Λ without reconstruction. The error in the sixth case was generated by adding capacitance in parallel to the resistance. We added a constant capacitance of $i\omega c = .8$ to each γ, but ignored the complex component of the result. The first five cases were rounded off to two significant digits. The latter technique produces error that is considerably more uniform than adding capacitances: the fractional error, E_λ/λ, is uniformly of magnitude 10^{-1}. The fractional error generated by the capacitance varied from 10^{-4} to 10^{-1}.

Constant γ's

σ_γ		σ_λ	
8.067	0.56236	8.067	0.0444216
8.241	0.56236	8.241	0.0444216
1.115	0.58121	8.183	0.04651382
3.177	0.58121	8.185	0.04651382
7.115	0.58121	8.178	0.06244455
1.153	0.60483	8.356	0.06248883
3.043	0.60483	8.112	0.07936929
8.178	0.62694	8.164	0.07936929
8.356	0.62694	6.034	0.1000759
8.112	0.65986	6.121	0.1000759
	0.8488		

γ varies from .3 to 21

σ_γ		σ_λ	
7.009	21.432	4.009	0.1786953
1.153	21.541	1.009	0.1791082
4.009	21.929	8.183	0.1794151
1.009	21.993	3.084	0.189903
8.007	22.436	7.009	0.1908004
6.080	22.634	6.084	0.1971246
8.183	24.744	6.168	0.2233695
3.084	25.413	2.169	0.2315341
3.086	26.359	8.265	0.3097188
6.130	27.825	6.121	0.3418671
	54.774		

γ varies from 1 to 9

σ_γ		σ_λ	
7.131	2.0453	8.183	0.1544017
7.128	2.4048	1.102	0.1628735
1.115	2.4563	8.164	0.1744031
7.115	2.4575	1.115	0.1784148
1.131	2.4771	7.115	0.185613
8.331	2.4869	8.178	0.1918926
1.128	2.6486	3.018	0.1923341
6.104	2.7259	6.104	0.2156397
2.115	2.7816	6.084	0.2381893
8.178	2.8339	2.102	0.2443825
	2.0299		

γ varies from 1 to 49

σ_γ		σ_λ	
3.162	8.8655	6.034	0.3458198
3.043	9.6078	6.045	0.37631
4.179	9.9606	6.121	0.5260968
1.167	10.095	3.043	0.5925738
6.034	10.354	3.162	0.6009474
8.260	10.412	6.138	0.6748162
6.121	10.801	6.153	0.7278545
2.045	10.843	1.045	0.7698654
1.078	11.145	2.045	0.7754161
8.067	11.237	8.067	0.798963
	16.248		

γ varies from .2 to 2			
σ_γ		σ_λ	
1.092	0.097724	1.092	0.02103216
3.129	0.097724	3.129	0.02103216
7.092	0.097724	7.092	0.02103216
5.000	0.17907	8.112	0.02726289
5.179	0.17907	8.185	0.02726289
8.018	0.1996	8.183	0.02778118
8.309	0.1996	1.009	0.03044748
1.009	0.21731	3.177	0.03044748
1.011	0.21731	4.009	0.03044748
3.056	0.21731	7.009	0.03044748
	0.30146		

γ varies from .2 to 2			
σ_γ		σ_λ	
8.158	3.081	6.029	0.3738372
8.265	3.081	6.045	0.3738372
2.108	4.3631	1.102	0.4243205
1.102	4.3632	3.151	0.4243205
3.151	4.3632	8.164	0.4243212
8.164	4.3632	8.241	0.4243212
8.241	4.3632	2.108	0.4243217
1.115	4.755	6.104	0.4400162
3.043	4.755	6.121	0.4400162
7.115	4.755	6.084	0.4446607
	0.38599		

In all of the cases we examined, we found that choosing a $\tilde{\Lambda}$ close to the original Λ was likely to yield a good $\tilde{\gamma}$ compared to the other reconstructions. We were not able to pick out the $\tilde{\Lambda}$ which produced the best $tilde\gamma$. Some of the "good" $\tilde{\Lambda}$'s produced average $tilde\gamma$'s (Halfway down the list sorted by distance). Furthermore, in some cases better results were obtained by directly calculating γ from the inaccurate Λ that could be obtained through any reconstruction. The third table (above) demonstrates this phenomena. We were unable to characterize the cases for which this occurs.

References

[1] E. B. Curtis and J. A. Morrow, *The Dirichlet to Neumann Map for a Resistor Network*

Appendices

A The FORTRAN Code LAMFILL

A.1 LAMFILL.F

```
C  This is the main section of the lambda reconstruction program
C  LAMFILL.F written by M. A. Myjak, 8-10-90 for the REU program.
      double precision ALPHA(2,10,36),LAMBDA(36,36),POINTS(36,36)
      implicit double precision (a-h, o-z)
      LOGICAL CHGPNTS
      NMAX=9
C  Call subroutine LAMIN.F to read in initial points in the lambda
C  matrix from which the reconstruction will proceed.
      CALL LAMIN(N,LAMBDA,POINTS,NFILLED,NDFILL)
      WRITE(12,*) 'This is an order',N,'system.'
      WRITE(12,*)
      WRITE(12,*) 'Number of entries filled from input:'
      WRITE(12,*) '                     DIAGONAL: ',NDFILL
      WRITE(12,*) '                 OFF-DIAGONAL: ',NFILLED
      WRITE(12,*)
C  Initialize various variables.
      NPNTSMD = (4*N)**2-4*N
      KMIN = 2
      KMAX = N+1
      K=KMIN
C  Initialize ALPHA
      DO 5 M=1,2
         DO 10 I=1,NMAX+1
            DO 20 J=1,4*NMAX
               ALPHA(M,I,J) = 0.0
 20         CONTINUE
 10      CONTINUE
 5    CONTINUE
C  Here begins the bulk of the main program.
 40   CONTINUE
C  Call solving-for-alphas routine
      CALL ALPHSOL(K,POINTS,ALPHA,N,NMAX)
C  Call solving-for-lambdas routine
      CALL PNTSSOL(K,POINTS,CHGPNTS,ALPHA,N,NFILLED)
      IF(NFILLED.GE.NPNTSMD) GOTO 50
      IF(CHGPNTS) THEN
         K=KMIN
         GOTO 40
      ELSE
         IF(K.LT.KMAX) THEN
            K=K+1
            GOTO 40
         ELSEIF(K.EQ.KMAX) THEN
            WRITE(12,*) 'Algorithm went as far as possible and'
            WRITE(12,*) 'LAMBDA NOT COMPLETE !'
            CALL DIAGSOL(POINTS,N,NDFILL)
            GOTO 100
         ENDIF
      ENDIF
C  Call solving for lambdas on the diagonal routine
 50   CALL DIAGSOL(POINTS,N,NDFILL)
      WRITE(12,*) 'EUREKA!  The LAMBDA is COMPLETE!'
C  Call output routine
 100  WRITE(12,*)
      WRITE(12,*) 'Elements known after reconstruction:'
      WRITE(12,*) '                     DIAGONAL: ',NDFILL
      WRITE(12,*) '                 OFF-DIAGONAL: ',NFILLED
      WRITE(12,*) '          Total elements known: ', NDFILL+NFILLED
      WRITE(12,*)
      CALL LAMOUT(N,LAMBDA,POINTS)
      END
```

A.2 LAMIN.F

```
C  This is SUBROUTINE LAMIN.F, the input subroutine for the main
C  program LAMFILL.F
      SUBROUTINE LAMIN(N,LAMBDA,POINTS,NFILLED,NDFILL)
      double precision LAMBDA(36,36),POINTS(36,36)
      implicit double precision (a-h, o-z)
      NFILLED = 0
      NDFILL = 0
      READ(11,*) N
C  Read initial POINTS values.
      DO 10 K = 1,4*N
         DO 12 L = 1,4*N
            READ(11,*) POINTS(K,L)
            IF(POINTS(K,L).NE.0.0) THEN
               IF(K.NE.L) THEN
                  NFILLED = NFILLED + 1
               ELSE
                  NDFILL = NDFILL + 1
               ENDIF
            ENDIF
 12      CONTINUE
         READ(11,*)
 10   CONTINUE
      DO 20 K = 1,4*N
         DO 30 L = 1,4*N
            IF(K.NE.L) THEN
               IF(POINTS(K,L).NE.0.0) THEN
                  IF(POINTS(L,K).EQ.0.0) THEN
                     POINTS(L,K) = POINTS(K,L)
                     NFILLED = NFILLED + 1
                  ENDIF
               ENDIF
            ENDIF
 30      CONTINUE
 20   CONTINUE
C  Read complete lambda matrix, for comparison at the end.
      read(10,*)
      do 13 k=1,4*n
         do 14 l=1,4*n
            read(10,*) lambda(k,l)
 14      continue
         read(10,*)
 13   continue
      RETURN
      END
```

A.3 ALPHSOL.F

```
C  This is SUBROUTINE ALPHSOL.F; part of the LAMFILL.F code.
      SUBROUTINE ALPHSOL(K,POINTS,ALPHA,N,NMAX)
      double precision POINTS(36,36),ALPHA(2,10,36)
      double precision A(36,36),B(36)
      implicit double precision (a-h, o-z)
      LOGICAL CHGALPH
      NEQREQ = K-1
      DO 5 M=1,2
         IF((M.EQ.2).AND.(K.EQ.2)) GOTO 5
         DO 10 J=1,4
C  If the ALPHAs are known, the loop advances...
            J0=MOD((1+(J-1)*N+(M-1)*(4*N-K+1)),(4*N))
            IF(J0.EQ.0) J0=4*N
            IF(ALPHA(M,K,J0).NE.0.0) GOTO 10
C  Ends loop advancing check.
C
C  Now the code looks for at least K-1 equations to solve for the
C  ALPHAs.  Since the coefficients of the ALPHAs and also the B-vector
C  itself are lambda values, the corresponding entries in the POINTS
C  array must be nonzero in order for the equation to be useful.
            IROW = 1
            DO 20 I=1,4*N-2*(K-1)
```

```
               I0=MOD((I+(K-1)+(J-1)*N),(4*N))
               IF(I0.EQ.0) I0=4*N
               I1=I0
               J1=MOD(((J-1)*N+K+4*N-2*(K-1)+(M-1)*(2*K-3)),(4*N))
               IF(J1.EQ.0) J1=4*N
               IF(POINTS(I1,J1).EQ.0.0) GOTO 20
               DO 30 L=1,K-1
                  J0=MOD((L+(J-1)*N+(M-1)*(4*N-K+1)),(4*N))
                  IF(J0.EQ.0) J0=4*N
                  IF(POINTS(I0,J0).EQ.0.0) GOTO 20
                  A(IROW,L) = POINTS(I0,J0)
 30            CONTINUE
               B(IROW)=-POINTS(I1,J1)
               IROW=IROW+1
 20         CONTINUE
            IROW=IROW-1
C  If there are not enough equations, the loop advances.
            IF(IROW.LT.NEQREQ) GOTO 10
C
C  EQCHEQ generates combinations of the IROW possible equations
C  found above and sends them to a linear solving routine.
            CALL EQCHEQ(A,NMAX,K,B,IROW,CHGALPH)
            IF(.NOT.CHGALPH) GOTO 10
C  If any of the "alphas-to-be" are zero (as can happen if error is
C  introduced into the original reconstruction parameters) then the
C  alphas are not stored.  Otherwise, a division by zero error
C  occurs.
            DO 60 L=1,K-1
               IF(B(L).EQ.0.0) GOTO 10
 60         CONTINUE
C  Store newly found alphas.
            DO 50 L=1,K-1
               J0=MOD((L+(J-1)*N+(M-1)*(4*N-K+1)),(4*N))
               IF(J0.EQ.0) J0=4*N
               ALPHA(M,K,J0)=B(L)
 50         CONTINUE
 10      CONTINUE
 5    CONTINUE
      RETURN
      END
```

A.4 EQCHEQ.F

```
C  This is SUBROUTINE EQCHEQ, part of LAMFILL.F
C  This was originally a routine to generate all possible comb.
C  of MXTOT things taken N at a time.  Here is it used to try
C  to find a set of equations which can be solved for the
C  ALPHAs.  The old code, excluding declarations, appears
C  in sections I and II.
      SUBROUTINE EQCHEQ(A,NMAX,K2,B,IROW,CHGALPH)
      implicit double precision (a-h, o-z)
      LOGICAL L, CHGALPH
      INTEGER V(20),X(20),MXNUM(252,10)
      INTEGER Y,Z,CNT,CNT2,IWORK(36)
      double precision work(36)
      double precision A(36,36), B(36), TEMPA(36,36), TEMPB(36)
      chgalph = .false.
      IF(K2.EQ.2) GOTO 470
      MXTOT=IROW
      N=K2-1
      DO 400 II=1,MXTOT
         DO 410 JJ=1,N
            TEMPA(II,JJ)=A(II,JJ)
 410     CONTINUE
         TEMPB(II)=B(II)
 400  CONTINUE
C  Here begins section I of the original combinations code.
      I=N-1
      M=FACT(MXTOT)/FACT(MXTOT-N)/FACT(N)
      V(1)=1
```

```
      J=N-2
      IF(J.LT.1) GOTO 43
      J1=-1
      DO 42 K1=1,J
         V(K1+1)=J1
 42   J1=J1-1
 43   Y=1
      Z=1
      DO 30 L1=1,N
         X(L1)=L1
 30   CONTINUE
      GOTO 10
 9    Y=Y+1
      Z=1
 10   continue
      DO 32 L2=1,N
         MXNUM(Y,L2)=X(L2)
 32   CONTINUE
C  Ends section I.
      DO 430 II=1,N
         DO 440 JJ=1,N
            A(II,JJ)=TEMPA(MXNUM(Y,II),JJ)
 440     CONTINUE
         B(II)=TEMPB(MXNUM(Y,II))
 430  CONTINUE
 470  CALL DGEFS(A,NMAX*4,(K2-1),B,1,IND,WORK,IWORK)
C  IND=-4 is assumed to imply that for the given combination of
C  equations, some subset contains equations which are linear
C  comb. of the other - like you can't have 2 eq. from a corner.
C  Also, if IND=-10, another system is looked for.  If
C  either IND=-4 or IND=-10, the next combination in the set
C  of possible equations is generated.
      IF((IND.EQ.-4).OR.(IND.EQ.-10)) THEN
         GOTO 100
      ELSE
         CHGALPH=.TRUE.
         GOTO 20
      ENDIF
C  The remaining code, section II, is original combinations
C  code.
 100  CONTINUE
      X(N)=X(N)+1
      IF(Y.EQ.M) GOTO 20
      CNT=0
      N1=N
      DO 40 K=1,I
         L=X(N1).EQ.(MXTOT+V(K))
         IF(.NOT.L) GOTO 50
         CNT=K
         IF(CNT.EQ.I.AND.L) GOTO 50
         N1=N1-1
 40   CONTINUE
 50   IF(CNT.EQ.0) GOTO 9
      CNT2=CNT
      X(N-CNT)=X(N-CNT)+1
      DO 200 J2=1,CNT2
         X(N-CNT+1)=X(N-CNT)+1
         CNT=CNT-1
         IF(CNT.EQ.0) GOTO 9
 200  CONTINUE
 20   CONTINUE
      RETURN
      END
      FUNCTION FACT(N)
      NPROD=1
      DO 300 J3=1,N
         NPROD=NPROD*J3
 300  CONTINUE
      FACT=NPROD
      RETURN
      END
```

A.5 PNTSSOL.F

```
C  This is SUBROUTINE PNTSSOL.F; it is part of LAMFILL.F.
      SUBROUTINE PNTSSOL(K,POINTS,CHGPNTS,ALPHA,N,NFILLED)
      double precision POINTS(36,36),ALPHA(2,10,36)
      implicit double precision (a-h, o-z)
      LOGICAL CHGPNTS
      CHGPNTS = .FALSE.
C  Solve for new lambdas using the ALPHA array.
      DO 5 M=1,2
         if((m.eq.2).and.(k.eq.2)) goto 5
         DO 50 J=1,4
            DO 60 I=1,4*N-2*(K-1)
               J0=MOD((1+(J-1)*N+(M-1)*(4*N-K+1)),(4*N))
               IF(J0.EQ.0) J0=4*N
               IF(ALPHA(M,K,J0).EQ.0.0) GOTO 50
               I0=MOD((I+(K-1)+(J-1)*N),(4*N))
               IF(I0.EQ.0) I0=4*N
               I1=I0
               J1=MOD(((J-1)*N+K+4*N-2*(K-1)+(M-1)*(2*K-3)),(4*N))
               IF(J1.EQ.0) J1=4*N
               IF(POINTS(I1,J1).EQ.0.0) THEN
                  NZ1=1
               ELSE
                  NZ1=0
               ENDIF
               NZ0=0
               DO 70 L=1,K-1
                  J0=MOD((L+(J-1)*N+(M-1)*(4*N-K+1)),(4*N))
                  IF(J0.EQ.0) J0=4*N
                  IF(POINTS(I0,J0).EQ.0.0) NZ0=NZ0+1
 70            CONTINUE
               NZSUM = NZ0 + NZ1
               IF((NZSUM.EQ.K).OR.(NZSUM.EQ.0)) GOTO 60
               IF((NZ1.EQ.1).AND.(NZ0.EQ.0)) THEN
                  DO 80 L=1,K-1
                     J0=MOD((L+(J-1)*N+(M-1)*(4*N-K+1)),(4*N))
                     IF(J0.EQ.0) J0=4*N
                     POINTS(I1,J1) = POINTS(I1,J1)-POINTS(I0,J0)*ALPHA(M,K,J0)
                     POINTS(J1,I1) = POINTS(I1,J1)
 80               CONTINUE
                  NFILLED = NFILLED + 2
                  CHGPNTS = .TRUE.
               ENDIF
               IF((NZ1.EQ.0).AND.(NZ0.EQ.1)) THEN
                  SUM = -POINTS(I1,J1)
                  DO 90 L=1,K-1
                     J0=MOD((L+(J-1)*N+(M-1)*(4*N-K+1)),(4*N))
                     IF(J0.EQ.0) J0=4*N
                     IF(POINTS(I0,J0).EQ.0.0) THEN
                        IFLAG = I0
                        JFLAG = J0
                        GOTO 90
                     ENDIF
                     SUM = SUM - POINTS(I0,J0)*ALPHA(M,K,J0)
 90               CONTINUE
                  POINTS(IFLAG,JFLAG) = SUM/ALPHA(M,K,JFLAG)
                  POINTS(JFLAG,IFLAG) = POINTS(IFLAG,JFLAG)
                  NFILLED = NFILLED + 2
                  CHGPNTS = .TRUE.
               ENDIF
 60         CONTINUE
 50      CONTINUE
 5    CONTINUE
 100  RETURN
      END
```

A.6 DIAGSOL.F

```
C  This is SUBROUTINE DIAGSOL.F, the code which finds diagonal
C  entries of the lambda matrix for the main program LAMFILL.F
```

```
      SUBROUTINE DIAGSOL(POINTS,N,NDFILL)
      double precision POINTS(36,36)
      implicit double precision (a-h, o-z)
      DO 10 I=1,4*N
         IF(POINTS(I,I).NE.0.0) GOTO 10
         SUM = 0.0
         DO 20 J=1,4*N
            IF(I.NE.J) THEN
               IF(POINTS(I,J).EQ.0.0) GOTO 10
               SUM = SUM - POINTS(I,J)
            ENDIF
 20      CONTINUE
         POINTS(I,I) = SUM
         NDFILL = NDFILL + 1
 10   CONTINUE
      RETURN
      END
```

A.7 LAMOUT.F

```
C  This is SUBROUTINE LAMOUT.F; it is part of the LAMFILL.F code.
C  LAMOUT.F is the output subroutine for LAMFILL.F.
      SUBROUTINE LAMOUT(N,LAMBDA,POINTS)
      double precision LAMBDA(36,36),POINTS(36,36),DIFF(36,36)
      implicit double precision (a-h, o-z)
      WRITE(12,*) 'Original Lambda:    Reconstructed Lambda:      Differ
     1ence'
      WRITE(16,*) N
      DO 10 I=1,4*N
         DO 20 J=1,4*N
            DIFF(I,J) = ABS(LAMBDA(I,J)-POINTS(I,J))
 100        FORMAT(3(1PD20.13,2X))
            WRITE(12,100) LAMBDA(I,J),POINTS(I,J),DIFF(I,J)
            WRITE(16,*) POINTS(I,J)
 20      CONTINUE
         write(12,*)
         write(16,*)
 10   CONTINUE
      RETURN
      END
```

B Sample Input and Output Files for LAMFILL.F

B.1 Part of a Sample fort.10 Input File

```
  3
  0.70089285714286
  -9.8214285714286d-02
  -3.1250000000000d-02
  -3.1250000000000d-02
  -2.6785714285714d-02
  -1.3392857142857d-02
  -1.3392857142857d-02
  -2.6785714285714d-02
  -3.1250000000000d-02
  -3.1250000000000d-02
  -9.8214285714286d-02
 -0.29910714285714

  -9.8214285714286d-02
  0.66964285714286
  -9.8214285714286d-02
  -9.8214285714286d-02
  -6.2500000000000d-02
  -2.6785714285714d-02
  -2.6785714285714d-02
  -4.4642857142857d-02
  -2.6785714285714d-02
  -2.6785714285714d-02
  -6.2500000000000d-02
  -9.8214285714286d-02

  -3.1250000000000d-02
```

B.2 Part of a Sample fort.11 Input File

```
3
 0
 0
 0
 0
 0
-1.3392857142857d-02
 0
 0
 0
 0
 0
 0

 0
 0
 0
 0
-6.2500000000000d-02
-2.6785714285714d-02
 0
 0
 0
 0
 0
 0

 0
```

B.3 Part of a Sample fort.12 Output File — 3×3 analog of template 2.116 shown in figure 5

```
 This is an order  3  system.
```

```
  Number of entries filled from input:
                       DIAGONAL:   0
                   OFF-DIAGONAL:   48

  Algorithm went as far as possible and
  LAMBDA NOT COMPLETE!

  Elements known after reconstruction:
                       DIAGONAL:   8
                   OFF-DIAGONAL:   128
             Total elements known:   136

   Original Lambda:    Reconstructed Lambda:      Difference
 7.0089285714286d-01   0.             d+00   7.0089285714286d-01
-9.8214285714286d-02  -9.8214285714185d-02   1.0126621763362d-13
-3.1250000000000d-02  -3.1250000000000d-02   0.              d+00
-3.1250000000000d-02  -3.1250000000000d-02   0.              d+00
-2.6785714285714d-02  -2.6785714285714d-02   0.              d+00
-1.3392857142857d-02  -1.3392857142857d-02   0.              d+00
-1.3392857142857d-02  -1.3392857142857d-02   0.              d+00
-2.6785714285714d-02  -2.6785714285714d-02   0.              d+00
-3.1250000000000d-02  -3.1250000000000d-02   0.              d+00
-3.1250000000000d-02  -3.1250000000000d-02   0.              d+00
-9.8214285714286d-02  -9.8214285714241d-02   4.5324854980322d-14
-2.9910714285714d-01   0.             d+00   2.9910714285714d-01
```

B.4 Part of a Sample fort.12 Output File — template 2.116 shown in figure 5

```
  This is an order  6  system.

  Number of entries filled from input:
                       DIAGONAL:   0
                   OFF-DIAGONAL:   168

  EUREKA!  The LAMBDA is COMPLETE!

  Elements known after reconstruction:
                       DIAGONAL:   24
                   OFF-DIAGONAL:   552
             Total elements known:   576

   Original Lambda:    Reconstructed Lambda:      Difference
 6.9797734342642d-01   3.4311888580228d-01   3.5485845762414d-01
-1.0404531314716d-01  -4.7217140590301d-02   5.6828172556859d-02
-4.1079910466438d-02  -3.0256410250892d-02   1.0823500215546d-02
-1.8162270768526d-02  -1.8162289460835d-02   1.8692309384366d-08
-8.4246146796267d-03  -8.4246146111234d-03   6.8503329744840d-11
-3.4103827207275d-03  -3.4103827207275d-03   0.              d+00
-3.4103827207275d-03  -3.4103827207275d-03   0.              d+00
-5.2169162032835d-03  -5.2169162032835d-03   0.              d+00
-5.3314768631529d-03  -5.3314768631529d-03   0.              d+00
-4.3918591432625d-03  -4.3918591432625d-03   0.              d+00
-3.0028111423460d-03  -3.0028111423460d-03   0.              d+00
-1.5014055711730d-03  -1.5014055711730d-03   0.              d+00
-1.5014055711730d-03  -1.5014055711730d-03   0.              d+00
-3.0028111423460d-03  -3.0028111423460d-03   0.              d+00
```

B.5 Part of a Sample fort.12 Output File — template 2.002 shown in figure 6

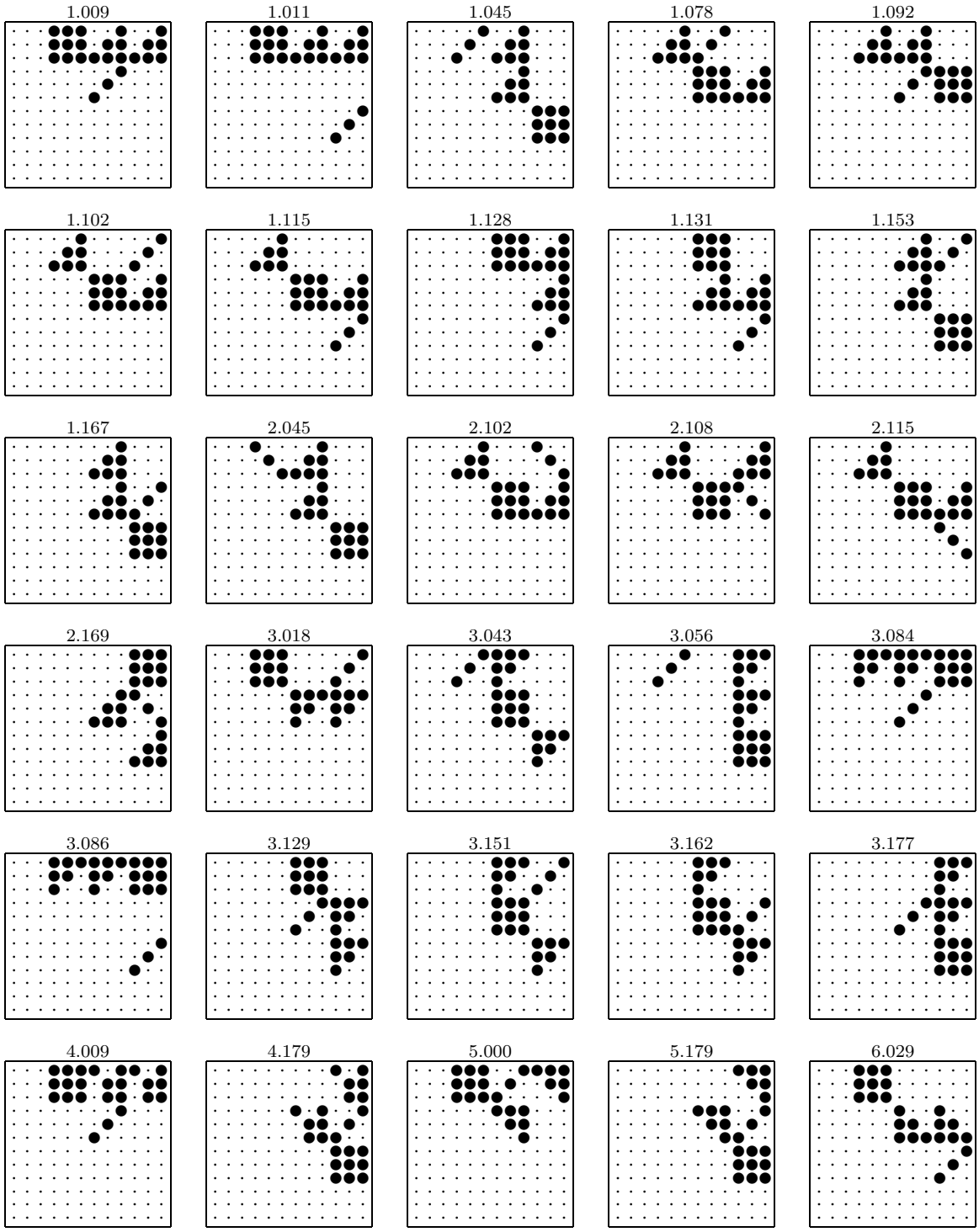
```
  This is an order  3  system.

  Number of entries filled from input:
                       DIAGONAL:   0
```

```
                    OFF-DIAGONAL:   48

  EUREKA!  The LAMBDA is COMPLETE!

  Elements known after reconstruction:
                         DIAGONAL:   12
                    OFF-DIAGONAL:   132
             Total elements known:   144

  Original Lambda:    Reconstructed Lambda:      Difference
 7.0089285714286d-01   7.0089285714281d-01   5.3179682879545d-14
-9.8214285714286d-02  -9.8214285714286d-02   0.              d+00
-3.1250000000000d-02  -3.1250000000000d-02   0.              d+00
-3.1250000000000d-02  -3.1250000000000d-02   0.              d+00
-2.6785714285714d-02  -2.6785714285714d-02   0.              d+00
-1.3392857142857d-02  -1.3392857142857d-02   0.              d+00
-1.3392857142857d-02  -1.3392857142857d-02   0.              d+00
-2.6785714285714d-02  -2.6785714285714d-02   0.              d+00
-3.1250000000000d-02  -3.1250000000000d-02   0.              d+00
-3.1250000000000d-02  -3.1250000000000d-02   0.              d+00
-9.8214285714286d-02  -9.8214285714239d-02   4.7337134212455d-14
-2.9910714285714d-01  -2.9910714285714d-01   0.              d+00

-9.8214285714286d-02  -9.8214285714286d-02   0.              d+00
```

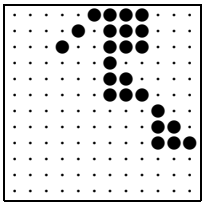
B.6 Part of a Sample fort.12 Output File — 6×6 analog of template 2.002 shown in figure 6

```
  This is an order  6  system.

  Number of entries filled from input:
                         DIAGONAL:   0
                    OFF-DIAGONAL:   168

  Algorithm went as far as possible and
  LAMBDA NOT COMPLETE!

  Elements known after reconstruction:
                         DIAGONAL:   2
                    OFF-DIAGONAL:   396
             Total elements known:   398

  Original Lambda:    Reconstructed Lambda:      Difference
 6.9797734342642d-01   0.              d+00   6.9797734342642d-01
-1.0404531314716d-01  -1.0404531314716d-01   0.              d+00
-4.1079910466438d-02  -4.1079910466438d-02   0.              d+00
-1.8162270768526d-02  -1.8162270768526d-02   0.              d+00
-8.4246146796267d-03  -8.4246146796267d-03   0.              d+00
-3.4103827207275d-03  -3.4103827207275d-03   0.              d+00
-3.4103827207275d-03  -3.4103827207275d-03   0.              d+00
-5.2169162032835d-03  -5.2169162032835d-03   0.              d+00
-5.3314768631529d-03  -5.3314768631529d-03   0.              d+00
-4.3918591432625d-03  -4.3918591432625d-03   0.              d+00
-3.0028111423460d-03  -3.0028111423460d-03   0.              d+00
-1.5014055711730d-03  -1.5014055711730d-03   0.              d+00
-1.5014055711730d-03  -1.5014055711730d-03   0.              d+00
```

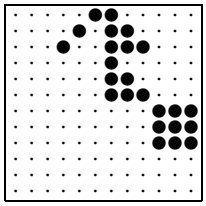
C Some Successful Parameter Sets



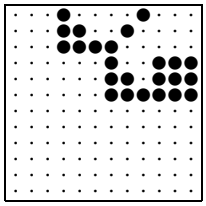
6.034



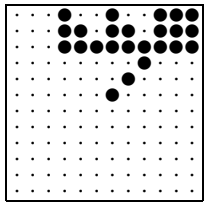
6.045



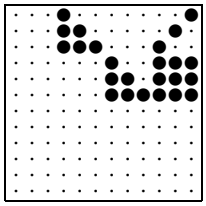
6.080



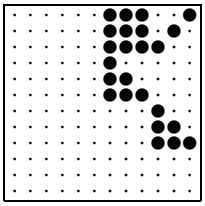
6.084



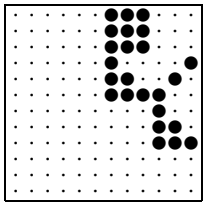
6.104



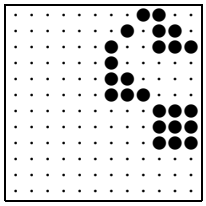
6.121



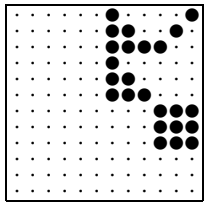
6.130



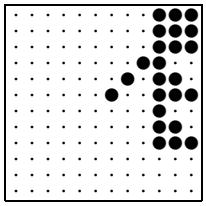
6.138



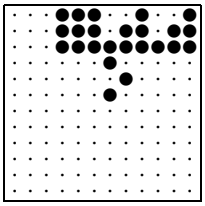
6.153



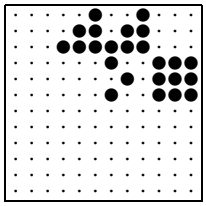
6.168



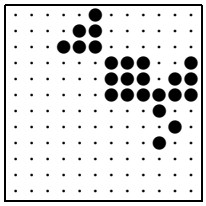
7.009



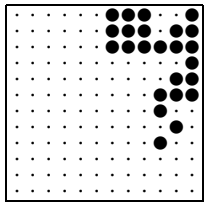
7.092



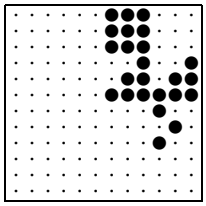
7.115



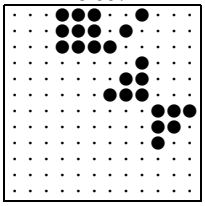
7.128



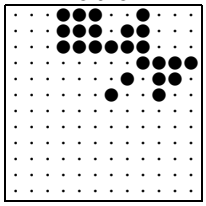
7.131



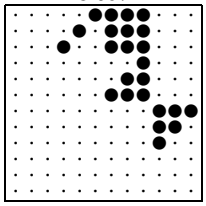
8.007



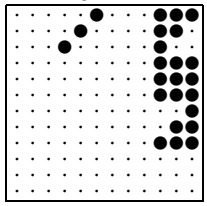
8.018



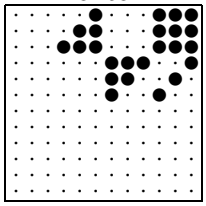
8.067



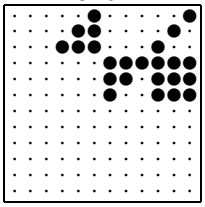
8.112



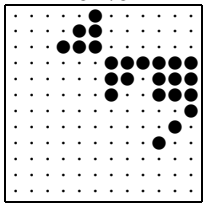
8.158



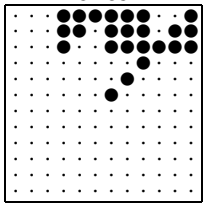
8.164



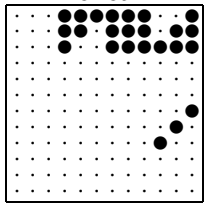
8.178



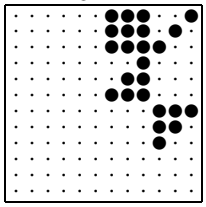
8.183



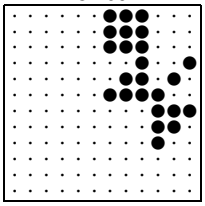
8.185



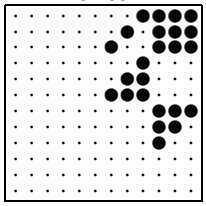
8.241



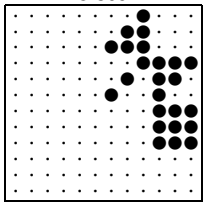
8.260



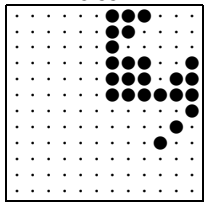
8.265



8.309



8.331



8.356