# Program 1

```
      program lambda

* variable declarations

      integer m, n, r, e, s, i, j, k, d, adim, job, info, ipvt(321)
      double precision g(361,361)
      double precision xK(321,321)
      double precision u(321), b(321)
      double precision A(40,40)
      double precision v(40)


* variable initializations

      adim = 321
      job = 0


* reading in m, n, and gamma values from a file

      print*,'Enter the number of the desired external file.'
      print*
      read (*,*) d
      read (d,*) m
      read (d,*) n
      do 10 i=1, (m+1)*n + 1
        do 20 j=1,(m+1)*n + 1
          read (d,*) g(i,j)
20      continue
10    continue


* creating Kirchhoff matrix K (note r is row # and e is entry (column) #)

      do 30 r=1, m*n + 1
        do 40 e=1, m*n + 1

* first n rows
          if (r.le.n) then

            if (r.eq.e) then
              if (r.eq.1) then
                xK(r,e) = g(n+r,r)+g(n+r,2*n+r)+g(n+r,n+r+1)+g(n+r,2*n)
              else
                if (r.eq.n) then
                  xK(r,e) = g(n+r,r)+g(n+r,n+1)+g(n+r,n+r-1)
                else
                  xK(r,e) = g(n+r,r)+g(n+r,n+r+1)+g(n+r,n+r-1)
                endif
                if (r.gt.(m-1)*n) then
                  xK(r,e) = xK(r,e)+g(n+r,(m+1)*n+1)
                else
                  xK(r,e) = xK(r,e)+g(n+r,2*n+r)
                endif
              endif
            endif

            if (r.eq.n) then
              if (e.eq.1) then
                xK(r,e) = -g(n+r,n+1)
              endif
            elseif (e.eq.r+1) then
                  xK(r,e) = -g(n+r,n+r+1)
            endif
```

```
          if (r.eq.1) then
            if (e.eq.n) then
              xK(r,e) = -g(n+r,2*n)
            endif
          elseif (e.eq.r-1) then
                  xK(r,e) = -g(n+r,n+r-1)
          endif

          if ((e.eq.m*n+1).and.(r.gt.(m-1)*n)) then
            xK(r,e) = -g(n+r,(m+1)*n+1)
          endif
          if ((e.eq.r+n).and.(r.le.(m-1)*n)) then
            xK(r,e) = -g(n+r,2*n+r)
          endif

        endif

* rows n+1 thru mn
        if (r.gt.n) then

          if (r.eq.e) then
           do 50 k=2,m
            if (r.eq.k*n+1) then
              xK(r,e) = g(n+r,r)+g(n+r,2*n+r)+g(n+r,n+r+1)+g(n+r,2*n+r-
            else
              if (r.eq.k*n) then
                xK(r,e) = g(n+r,r)+g(n+r,r+1)+g(n+r,n+r-1)
              else
                xK(r,e) = g(n+r,r)+g(n+r,n+r+1)+g(n+r,n+r-1)
              endif
              if (r.gt.(m-1)*n) then
                xK(r,e) = xK(r,e)+g(n+r,(m+1)*n+1)
              else
                xK(r,e) = xK(r,e)+g(n+r,2*n+r)
              endif
            endif
50         continue
          endif

         do 60 k=2,m
          if (r.eq.k*n) then
            if (e.eq.r-n+1) then
              xK(r,e) = -g(n+r,r+1)
            endif
          elseif (e.eq.r+1) then
                  xK(r,e) = -g(n+r,n+r+1)
          endif
60       continue

         do 70 k=2,m
          if (r.eq.k*n+1) then
            if (e.eq.r+n-1) then
              xK(r,e) = -g(n+r,2*n+r-1)
            endif
          elseif (e.eq.r-1) then
                  xK(r,e) = -g(n+r,n+r-1)
          endif
70       continue

          if ((e.eq.m*n+1).and.(r.gt.(m-1)*n).and.(r.lt.m*n+1)) then
            xK(r,e) = -g(n+r,(m+1)*n+1)
          endif
          if ((e.eq.r+n).and.(r.le.(m-1)*n)) then
            xK(r,e) = -g(n+r,2*n+r)
          endif
```

```
              if (e.eq.r-n) then
                xK(r,e) = -g(n+r,r)
              endif

* row mn+1
              if (r.eq.m*n+1) then
                if ((e.ge.(m-1)*n+1).and.(e.le.m*n)) then
                  xK(r,e) = -g((m+1)*n+1,e+n)
                endif
                if (e.eq.m*n+1) then
                  do 80 s=m*n+1,(m+1)*n
                    xK(r,e) = xK(r,e) + g((m+1)*n+1,s)
80                continue
                endif
              endif

            endif

40        continue
30      continue

* printing the Kirchhoff matrix

        print*
        print*,'The Kirchhoff Matrix is:'
        do 90 i=1,m*n+1
          print*
          print*,'*'
          print*
          do 100 j=1,m*n+1
            print*, xK(i,j)
100       continue
90      continue
        print*,'Enter a number to see lambda matrix.'
        read (*,*) s

* solving matrix equation Ku = b using Linpack subroutines DGEFA and DGESL
* and forming the lambda matrix A

        call DGEFA (xK, adim, (m*n+1), ipvt, info)

        do 110 i=1,n
          do 120 j=1,m*n+1
            if (j.eq.i) then
              b(j) = g(i,(i+n))
            else
              b(j) = 0
            endif
120       continue
          call DGESL (xK, adim, (m*n+1), ipvt, b, job)
          do 130 s=1,n
            if (s.eq.i) then
              A(i,s) = (1.d0-b(s))*(g(s,(s+n)))
            else
              A(i,s) = -(b(s))*(g(s,(s+n)))
            endif
130       continue
110     continue

* printing the lambda matrix

        print*
        print*
```

```
      print*,'The Lambda Matrix is:'
      do 140 i=1,n
        print*
        print*,'*'
        print*
        do 150 j=1,n
          print*, A(i,j)
150     continue
140   continue


      stop
      end
```

# Program 2

```
      program gamma

* variable declarations

      integer i,j,k,m,n
      double precision g(361,361)


* getting preliminary information from user

      print*
      print*,'Enter the number of circles.'
      read (*,*) m
      print*
      print*,'Enter the number of rays.'
      read (*,*) n
      write (1,*) m
      write (1,*) n


* loop to read all the gamma values

      do 10 i=1,(m+1)*n+1
        do 20 j=1,(m+1)*n+1

* conductors on the boundary

          if ((i.le.n).and.(j.eq.(i+n))) then
            print*,'Enter g(',i,',',j,').'
            read (*,*) g(i,j)
          endif

* interior circular conductors

          if ((i.gt.n).and.(i.le.(m+1)*n)) then
            do 30 k=1,(m+1)
              if (i.eq.k*n) then
                if (j.eq.(k-1)*n+1) then
                  print*,'Enter g(',i,',',j,').'
                  read (*,*) g(i,j)
                endif
                goto 40
              endif
30          continue
            if (j.eq.i+1) then
              print*,'Enter g(',i,',',j,').'
              read (*,*) g(i,j)
40          endif

* interior radial conductors

            if ((i.le.m*n).and.(j.eq.(i+n))) then
              print*,'Enter g(',i,',',j,').'
              read (*,*) g(i,j)
            endif
            if ((i.gt.m*n).and.(j.eq.(m+1)*n+1)) then
              print*,'Enter g(',i,',',j,').'
              read (*,*) g(i,j)
            endif

          endif

* setting permutations of the same pair equal

          g(j,i) = g(i,j)
```

```
20        continue
10      continue

* writing the gamma values to the external file

        do 50 i=1,(m+1)*n+1
          do 60 j=1,(m+1)*n+1
            write (1,*) g(i,j)
60        continue
50      continue


        stop
        end
```

# Program 3

```
      program lambda

* variable declarations

      integer m, n, r, e, s, i, j, k, adim, job, info, ipvt(321)
      double precision g(361,361)
      double precision xK(321,321)
      double precision u(321), b(321)
      double precision A(40,321)
      double precision v(40)


* variable initializations

      adim = 321
      job = 0


* introduction

      print*
      print*,'This program assumes a circular network of'
      print*,'form C_1(m,n), with one or more source currents'
      print*,'at interior nodes (all flowing in) and zero'
      print*,'potential at each boundary node.'
      print*
      print*,'The Kirchhoff matrix is computed in the same'
      print*,'manner as with a network having no interior'
      print*,'sources, but the so-called Lambda matrix is'
      print*,'a bit different.  In this program the Lambda'
      print*,'matrix represents the map from interior source'
      print*,'currents to boundary currents.  The dimensions'
      print*,'are n by m*n+1, and the ij entry is the current'
      print*,'which would flow out of boundary node i if there'
      print*,'were a unit source current at the jth interior'
      print*,'node and zero source current at all other interior'
      print*,'nodes.'
      print*
      print*,'For some configuration of interior source'
      print*,'currents, entered in an m*n+1 by 1 column'
      print*,'vector, the outflowing boundary currents are'
      print*,'obtained (in columnar form) by multiplying on'
      print*,'the left by the Lambda matrix.'
      print*


* reading in m, n, and gamma values from a file

      read (1,*) m
      read (1,*) n
      do 10 i=1, (m+1)*n + 1
        do 20 j=1,(m+1)*n + 1
          read (1,*) g(i,j)
20      continue
10    continue


* creating Kirchhoff matrix K (note r is row # and e is entry (column) #)

      do 30 r=1, m*n + 1
        do 40 e=1, m*n + 1

* first n rows
          if (r.le.n) then
```

```
            if (r.eq.e) then
              if (r.eq.1) then
                xK(r,e) = g(n+r,r)+g(n+r,2*n+r)+g(n+r,n+r+1)+g(n+r,2*n)
              else
                if (r.eq.n) then
                  xK(r,e) = g(n+r,r)+g(n+r,n+1)+g(n+r,n+r-1)
                else
                  xK(r,e) = g(n+r,r)+g(n+r,n+r+1)+g(n+r,n+r-1)
                endif
                if (r.gt.(m-1)*n) then
                  xK(r,e) = xK(r,e)+g(n+r,(m+1)*n+1)
                else
                  xK(r,e) = xK(r,e)+g(n+r,2*n+r)
                endif
              endif
            endif

            if (r.eq.n) then
              if (e.eq.1) then
                xK(r,e) = -g(n+r,n+1)
              endif
            elseif (e.eq.r+1) then
                    xK(r,e) = -g(n+r,n+r+1)
            endif

            if (r.eq.1) then
              if (e.eq.n) then
                xK(r,e) = -g(n+r,2*n)
              endif
            elseif (e.eq.r-1) then
                    xK(r,e) = -g(n+r,n+r-1)
            endif

            if ((e.eq.m*n+1).and.(r.gt.(m-1)*n)) then
              xK(r,e) = -g(n+r,(m+1)*n+1)
            endif
            if ((e.eq.r+n).and.(r.le.(m-1)*n)) then
              xK(r,e) = -g(n+r,2*n+r)
            endif

          endif

* rows n+1 thru mn
          if (r.gt.n) then

            if (r.eq.e) then
             do 50 k=2,m
              if (r.eq.k*n+1) then
                xK(r,e) = g(n+r,r)+g(n+r,2*n+r)+g(n+r,n+r+1)+g(n+r,2*n+r-
              else
                if (r.eq.k*n) then
                  xK(r,e) = g(n+r,r)+g(n+r,r+1)+g(n+r,n+r-1)
                else
                  xK(r,e) = g(n+r,r)+g(n+r,n+r+1)+g(n+r,n+r-1)
                endif
                if (r.gt.(m-1)*n) then
                  xK(r,e) = xK(r,e)+g(n+r,(m+1)*n+1)
                else
                  xK(r,e) = xK(r,e)+g(n+r,2*n+r)
                endif
              endif
50           continue
            endif

           do 60 k=2,m
            if (r.eq.k*n) then
```

```
            if (e.eq.r-n+1) then
              xK(r,e) = -g(n+r,r+1)
            endif
          elseif (e.eq.r+1) then
                  xK(r,e) = -g(n+r,n+r+1)
          endif
60       continue

         do 70 k=2,m
          if (r.eq.k*n+1) then
            if (e.eq.r+n-1) then
              xK(r,e) = -g(n+r,2*n+r-1)
            endif
          elseif (e.eq.r-1) then
                  xK(r,e) = -g(n+r,n+r-1)
          endif
70       continue

          if ((e.eq.m*n+1).and.(r.gt.(m-1)*n).and.(r.lt.m*n+1)) then
            xK(r,e) = -g(n+r,(m+1)*n+1)
          endif
          if ((e.eq.r+n).and.(r.le.(m-1)*n)) then
            xK(r,e) = -g(n+r,2*n+r)
          endif

          if (e.eq.r-n) then
            xK(r,e) = -g(n+r,r)
          endif

* row mn+1
          if (r.eq.m*n+1) then
            if ((e.ge.(m-1)*n+1).and.(e.le.m*n)) then
              xK(r,e) = -g((m+1)*n+1,e+n)
            endif
            if (e.eq.m*n+1) then
              do 80 s=m*n+1,(m+1)*n
                xK(r,e) = xK(r,e) + g((m+1)*n+1,s)
80            continue
            endif
          endif

        endif

40     continue
30    continue


* printing the Kirchhoff matrix

      print*
      print*,'The Kirchhoff Matrix is:'
      do 90 i=1,m*n+1
        print*
        print*,'*'
        print*
        do 100 j=1,m*n+1
          print*, xK(i,j)
100     continue
90    continue
      print*,'Enter a number to see lambda matrix.'
      read (*,*) s


* solving matrix equation Ku = b using Linpack subroutines DGEFA and DGESL
* and forming the lambda matrix A
```

```
      call DGEFA (xK, adim, (m*n+1), ipvt, info)

      do 110 i=1,m*n+1
        do 120 j=1,m*n+1
          if (j.eq.i) then
            b(j) = 1
          else
            b(j) = 0
          endif
120     continue
        call DGESL (xK, adim, (m*n+1), ipvt, b, job)
        do 130 s=1,n
          A(s,i) = -b(s)*g(s,s+n)
130     continue
110   continue

* printing the lambda matrix

      print*
      print*
      print*,'The Lambda Matrix is:'
      do 140 i=1,n
        print*
        print*,'*'
        print*
        do 150 j=1,m*n+1
          print*, A(i,j)
150     continue
140   continue

* writing the lambda matrix and potential at center node to an external fil

      do 160 i=1,n
        do 170 j=1,m*n+1
          write (2,*) A(i,j)
170     continue
160   continue
      write (2,*) b(m*n+1)


      stop
      end
```

# Program 4

```
      program inverse

* This program will take the entries of the lambda
* (mapping interior source currents to boundary
* currents) matrix and use them to solve for the
* values of gamma in a circular network with one
* circle and three rays.  In order to solve this
* network, it will also be assumed that the three
* boundary conductors have unit conductivity.


* VARIABLE DECLARATIONS

      integer f, i, j, ipvt(3), job, adim, n, info
      double precision b(3), u(3), t(3), A(3,4)
      double precision P(3,3), Q(3,3), R(3,3), L(3,3)
      double precision x(3), y(3), z(3)


* INTRODUCTORY MESSAGE

      print*
      print*,'This program is for m=1, n=3, unit boundary gammas,'
      print*,'with only the lambda matrix known.'
      print*


* READING IN THE LAMBDA MATRIX FROM AN EXTERNAL FILE

      print*,'Enter the number of the .fort file in'
      print*,'which the desired lambda matrix is stored.'
      read (*,*) f
      do 10 i=1,3
        do 20 j=1,4
          read (f,*) A(i,j)
20      continue
10    continue


* FIRST EXPERIMENT FOR RECOVERING CIRCULAR CONDUCTORS

* setting up the matrix Q

      do 30 i=1,2
        do 40 j=1,3
          Q(i,j) = -A(i,j)
40      continue
30    continue
      do 50 i=1,3
        Q(3,i) = -A(i,4)
50    continue


* solving Qi=b, where i is a vector of source currents at
* nodes 1,2,3 and b is a vector of potentials at nodes 1,2,4
* [Linpack routines DGEFA and DGESL are used]

      job = 0
      adim = 3
      n = 3
      call DGEFA (Q, adim, n, ipvt, info)
      b(1) = 1
      b(2) = 0.5
      b(3) = 1
      call DGESL (Q, adim, n, ipvt, b, job)
```

```
        do 100 i=1,3
          x(i) = b(i)
100     continue

* SECOND EXPERIMENT FOR RECOVERING CIRCULAR CONDUCTORS

* setting up the matrix R

        do 110 i=1,3
          R(1,i) = -A(2,i)
110     continue
        do 120 i=1,3
          R(2,i) = -A(3,i)
120     continue
        do 125 i=1,3
          R(3,i) = -A(i,4)
125     continue

* solving Ri=b, where i is a vector of source currents at
* nodes 1,2,3 and b is a vector of potentials at nodes 2,3,4
* [Linpack routines DGEFA and DGESL used]

        call DGEFA (R, adim, n, ipvt, info)
        b(1) = 1
        b(2) = 0.5
        b(3) = 1
        call DGESL (R, adim, n, ipvt, b, job)
        do 130 i=1,3
          y(i) = b(i)
130     continue

* THIRD EXPERIMENT FOR RECOVERING CIRCULAR CONDUCTORS

* setting up the matrix L

        do 150 i=1,3
          L(1,i) = -A(2,i)
150     continue
        do 160 i=1,3
          L(2,i) = -A(3,i)
160     continue
        do 170 i=1,3
          L(3,i) = -A(i,4)
170     continue

* solving Li=b, where i is a vector of source currents at
* nodes 1,2,3 and b is a vector of potentials at nodes 2,3,4
* [Linpack routines DGEFA and DGESL used]

        call DGEFA (L, adim, n, ipvt, info)
        b(1) = 0.5
        b(2) = 1
        b(3) = 1
        call DGESL (L, adim, n, ipvt, b, job)
        do 140 i=1,3
          z(i) = b(i)
140     continue

* OBTAINING THE UNKNOWN POTENTIAL IN EACH OF THE ABOVE
* CONFIGURATIONS

        u(1) = -(x(1)*A(3,1)+x(2)*A(3,2)+x(3)*A(3,3))
```

```
      u(2) = -(y(1)*A(1,1)+y(2)*A(1,2)+y(3)*A(1,3))
      u(3) = -(z(1)*A(1,1)+z(2)*A(1,2)+z(3)*A(1,3))

* RECOVERING THE CIRCULAR CONDUCTORS
* [THE THREE GAMMA VALUES WILL BE STORED IN b]

      P(1,1) = 0.5
      P(1,2) = 0
      P(1,3) = 1-u(1)
      P(2,1) = 1-u(2)
      P(2,2) = 0.5
      P(2,3) = 0
      P(3,1) = 0
      P(3,2) = 0.5
      P(3,3) = 1-u(3)

      call DGEFA (P, adim, n, ipvt, info)

      b(1) = x(1)-1
      b(2) = y(2)-1
      b(3) = z(3)-1
      call DGESL (P, adim, n, ipvt, b, job)

* RECOVERING THE RADIAL CONDUCTORS

      t(1) = (b(1)+0.5*b(3)-u(2)*(b(1)+b(3)+1))/u(2)-1
      t(2) = -(b(1)+u(1)*b(2)-0.5*(b(1)+b(2)+1))/0.5
      t(3) = (b(3)+0.5*b(2)-u(1)*(b(2)+b(3)+1))/u(1)-1

* PRINTING THE CONDUCTIVITIES

      print*
      print*,'Gamma of 1 and 2 is:',b(1)
      print*,'Gamma of 2 and 3 is:',b(2)
      print*,'Gamma of 1 and 3 is:',b(3)
      print*,'Gamma of 1 and 4 is:',t(1)
      print*,'Gamma of 2 and 4 is:',t(2)
      print*,'Gamma of 3 and 4 is:',t(3)


      stop
      end
```

# Program 5

```
      program inverse

* This program will take the entries of the alpha
* (mapping interior source currents to boundary
* currents) matrix and use them to solve for the
* values of gamma in a circular network with one
* circle and four rays.  In order to solve this
* network, it will also be assumed that the four
* boundary conductors have unit conductivity.

* last updated PM 8/5/92


* VARIABLE DECLARATIONS

      integer i, j, ipvt(3), job, adim, n, info
      double precision g(8), b(3), A(4,5), u, x
      double precision P(5,4), Q(3,3), R(3,3)


* INTRODUCTORY MESSAGE

      print*
      print*,'This program is for m=1, n=4, unit boundary gammas,'
      print*,'with only the alpha matrix known.'
      print*
      print*,'The alpha matrix is read from the file'
      print*,'"fort.2", and the recovered gamma values appear below.'


* READING IN THE ALPHA MATRIX FROM AN EXTERNAL FILE

      do 10 i=1,4
        do 20 j=1,5
          read (2,*) A(i,j)
20      continue
10    continue


* CREATING THE "BASE" MATRIX P

      do 15 i=1,4
        do 25 j=1,4
          P(i,j) = -A(i,j)
25      continue
15    continue
      do 35 j=1,4
        P(5,j) = -A(j,5)
35    continue


* FIRST EXPERIMENT FOR RECOVERING CIRCULAR CONDUCTORS

* setting up the matrix Q

      do 40 j=1,3
        Q(1,j) = P(1,j)
        Q(2,j) = P(4,j)
        Q(3,j) = P(5,j)
40    continue


* solving Qi=b, where i is a vector of source currents at
* nodes 1,2,3 and b is a vector of potentials at nodes 1,4,5
* [Linpack routines DGEFA and DGESL are used]
```

```
      job = 0
      adim = 3
      n = 3
      call DGEFA (Q, adim, n, ipvt, info)
      b(1) = 1
      b(2) = 1
      b(3) = 1
      call DGESL (Q, adim, n, ipvt, b, job)

* solving the first circular conductivity

      u = b(1)*P(2,1)+b(2)*P(2,2)+b(3)*P(2,3)
      g(1) = (b(1)-1)/(1-u)

* solving the third circular conductivity

      u = b(1)*P(3,1)+b(2)*P(3,2)+b(3)*P(3,3)
      g(3) = -1/(1-u)


* SECOND EXPERIMENT FOR RECOVERING CIRCULAR CONDUCTORS

* setting up the matrix R

      do 110 j=1,3
        R(1,j) = P(1,j+1)
        R(2,j) = P(2,j+1)
        R(3,j) = P(5,j+1)
110   continue


* solving Ri=b, where i is a vector of source currents at
* nodes 2,3,4 and b is a vector of potentials at nodes 1,2,5
* [Linpack routines DGEFA and DGESL used]

      call DGEFA (R, adim, n, ipvt, info)
      b(1) = 1
      b(2) = 1
      b(3) = 1
      call DGESL (R, adim, n, ipvt, b, job)

* solving the second circular conductivity

      u = b(1)*P(3,2)+b(2)*P(3,3)+b(3)*P(3,4)
      g(2) = (b(1)-1)/(1-u)

* solving the fourth circular conductivity

      u = b(1)*P(4,2)+b(2)*P(4,3)+b(3)*P(4,4)
      g(4) = -1/(1-u)


* RECOVERING THE RADIAL CONDUCTORS

      x = -(P(1,1)+g(1)*(P(1,1)-P(2,1))+g(4)*(P(1,1)-P(4,1))-1)
      g(5) = x/(P(1,1)-P(5,1))

      x = -(P(2,1)+g(1)*(P(2,1)-P(1,1))+g(2)*(P(2,1)-P(3,1)))
      g(6) = x/(P(2,1)-P(5,1))

      x = -(P(3,1)+g(2)*(P(3,1)-P(2,1))+g(3)*(P(3,1)-P(4,1)))
      g(7) = x/(P(3,1)-P(5,1))

      x = -(P(4,1)+g(4)*(P(4,1)-P(1,1))+g(3)*(P(4,1)-P(3,1)))
      g(8) = x/(P(4,1)-P(5,1))
```

```
* PRINTING THE CONDUCTIVITIES

      print*
      print*,'Gamma of 1 and 2 is:',g(1)
      print*,'Gamma of 2 and 3 is:',g(2)
      print*,'Gamma of 3 and 4 is:',g(3)
      print*,'Gamma of 1 and 4 is:',g(4)
      print*,'Gamma of 1 and 5 is:',g(5)
      print*,'Gamma of 2 and 5 is:',g(6)
      print*,'Gamma of 3 and 5 is:',g(7)
      print*,'Gamma of 4 and 5 is:',g(8)
      print*


      stop
      end
```