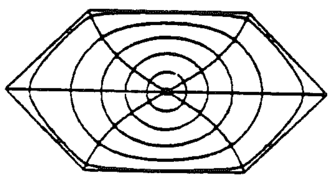
Numerical Analysis Report 89-2
Department of Mathematics, M.I.T.
January 1989

SCPACK USER'S GUIDE

Description of a collection of Fortran programs for Schwarz-Christoffel conformal mapping

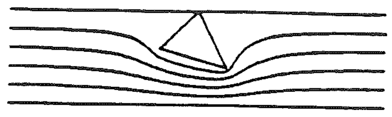
Lloyd N. Trefethen

Dept. of Mathematics, M.I.T.
Cambridge, MA 02139 USA
(617) 253-4986
lnt@math.mit.edu



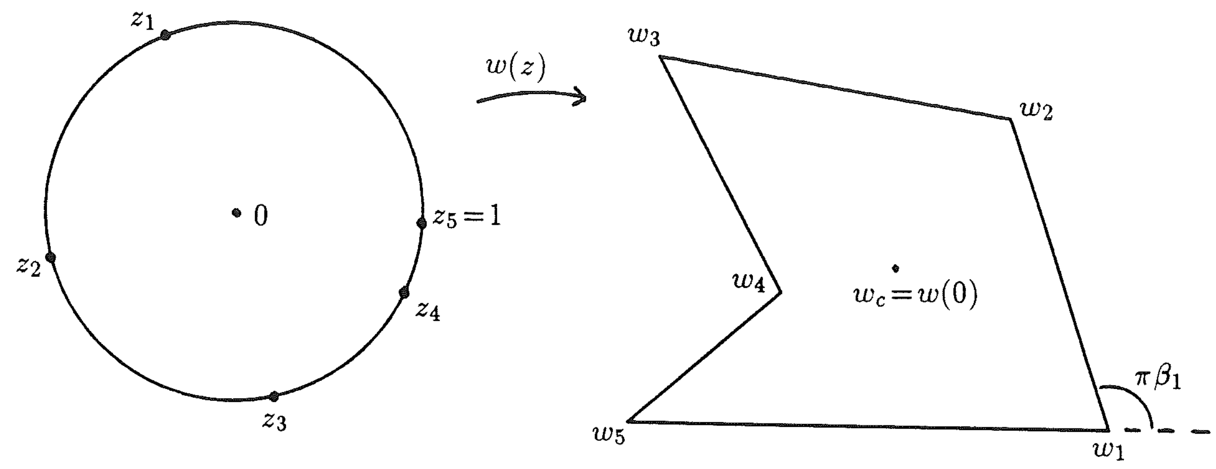
This user's guide is an update of an earlier (1983) ICASE internal report.
The SCPACK program itself has not changed.

Contents



1. Introduction

SCPACK is a collection of Fortran programs for computing the Schwarz-Christoffel transformation, a conformal map of the unit disk in the complex plane onto a polygon with vertices $w_1, \ldots, w_N$. Here is the geometry, illustrated for a simple polygon with $N = 5$:



And here is the Schwarz-Christoffel formula:

$$w(z) \;=\; w_c + C\int_0^z \prod_{k=1}^{N} (1 - z'/z_k)^{-\beta_k} dz', \qquad (*)$$

or in the notation of SCPACK variables,

$$\mathtt{W(Z)} \;=\; \mathtt{WC} + \mathtt{C}\int_0^{\mathtt{Z}} \prod_{\mathtt{K=1}}^{\mathtt{N}} (1 - \mathtt{Z}'/\mathtt{Z(K)})^{\mathrm{BETAM(K)}} \mathtt{dZ}'. \qquad (*)$$

There are three computational problems associated with $(*)$: *(1)* evaluation of the integral; *(2)* the "parameter problem" of determining the unknown "prevertices" z_k, which are the preimages on the unit circle of the vertices w_k; and *(3)* inversion of the map to compute values $z(w)$. SCPACK aims to deal with these problems efficiently, robustly, and conveniently. Given the vertices $\{w_k\}$ and corresponding angle parameters $-\beta_k$, SCPACK first determines parameters for compound Gauss-Jacobi quadrature in subroutine `QINIT`, and then solves the parameter problem in subroutine `SCSOLV`. After this the functions `WSC` and `ZSC` can be called to evaluate the forward and inverse maps $w(z)$ and $z(w)$.

Some of the vertices w_k may lie at infinity, so the "polygon" is really any simply-connected region in the plane whose boundary consists of a finite number of straight lines or line segments.

Schwarz-Christoffel maps can be applied to a variety of problems involving polygonal domains, such as: solving Laplace's equation with Dirichlet, Neumann, oblique, or mixed boundary conditions; solving Poisson's equation; determination of resistances, capacitances, and conformal moduli; finding eigenvalues of the Laplace operator; grid generation; approximation in the complex plane via interpolation in Fejér points; computing ideal free-streamline flows.

The algorithms of SCPACK are described along with some elementary applications in

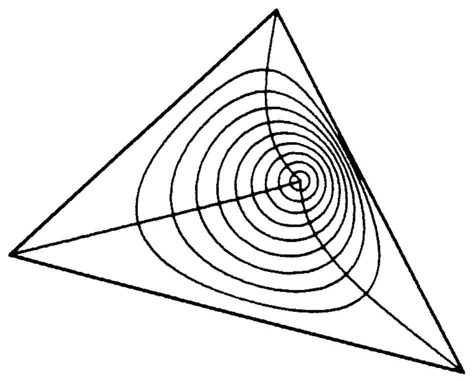
[1] L.N. Trefethen, "Numerical computation of the Schwarz-Christoffel transformation," *SIAM J. Sci. Stat. Comput. 1* (1980), 82–102.

A survey of more interesting applications of Schwarz-Christoffel mapping, and of variations of the standard Schwarz-Christoffel formula for other related conformal mapping problems, can be found in

[2] L.N. Trefethen, "Schwarz-Christoffel mapping in the 1980's," Numerical Analysis Report 89-1, Dept. of Mathematics, M.I.T., 1989 (available from L.N.T.).

Further references are given in Section 6.

The use of SCPACK is unrestricted, except that references to [1] and to this User's Guide should be included in any publications that make use of it. I would be pleased to receive reprints of such publications, or less formal descriptions of applications of SCPACK. I would also be grateful for suggestions of improvements in the program or its documentation.



2. How to obtain SCPACK

The easiest way to obtain SCPACK is by electronic mail via the "Netlib" facility created by Jack Dongarra (Argonne National Laboratory) and Eric Grosse (AT&T Bell Laboratories):

[3] J. J. Dongarra and E. Grosse, "Distribution of mathematical software via electronic mail," *Communications of the ACM 30* (1987), 403–407.

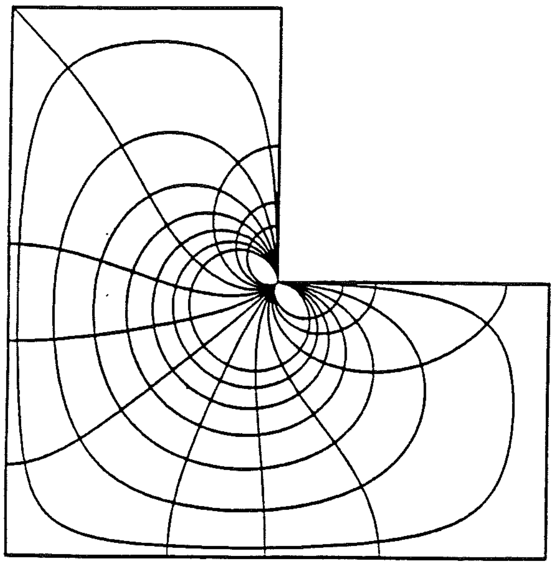
For example, send the following three-line e-mail message to `netlib@anl-mcs.arpa`:

```
send scpdbl from conformal
send sclibdbl from conformal
send index from conformal
```

If all goes well, you will receive three responses within a few minutes or hours consisting of SCPACK itself (`scpdbl`), three library routines required by SCPACK (`sclibdbl`, described in 3.4 below), and an up-to-date index of Netlib software for conformal mapping (`index`). Unfortunately, not much conformal mapping software is currently available in Netlib or anywhere else, although many good algorithms have been developed over the years.

If you cannot obtain SCPACK by computer mail, then contact me to arrange for a magnetic tape. Contact me also for copies of this User's Guide.

There are no charges associated with SCPACK or the User's Guide (and no guarantees either!). The User's Guide may be copied at will.



3. Description of SCPACK

3.1. Capabilities and speed

SCPACK contains routines to solve the "parameter problem" associated with the Schwarz-Christoffel map (subroutines `QINIT` and `SCSOLV`) and then to evaluate the map $w(z)$ (`WSC`) and its inverse $z(w)$ (`ZSC`). A typical computation on a Sun 3/50 Workstation requires a few seconds for a polygon with 6 or 7 vertices, or a few minutes for a polygon with 20 vertices; the total time for an N-vertex polygon is roughly proportional to N^3. See reference [1] for a more detailed discussion of execution speed.

The bottleneck in all SCPACK computations is a calculation of a complex logarithm within subroutine `ZPROD` (p. 34 of this User's Guide; see also Section 4.7). Some optimization here might speed up the package at particular installations.

For high-accuracy conformal mapping of a polygon, the Schwarz-Christoffel approach is extremely satisfactory because it handles the singularities at corners exactly and reduces the map to a finite number of parameters. On the other hand SCPACK is not recommended for mapping curved domains by means of polygonal approximations; the $O(N^3)$ running time and the introduction of spurious singularities make this approach frighteningly inefficient in some circumstances. Many superior methods exist for such problems; see the references in Section 6.

3.2. Fortran environment

SCPACK is written in a subset of Fortran 77 — almost. The exception is the use of complex double precision variables, which are not prescribed in the Fortran 77 standard but are provided by most compilers anyway.

All real numbers in SCPACK are double precision and all complex numbers are complex double precision. (A single precision version of SCPACK also exists, and can be obtained from Netlib, but because of the phenomenon of "crowding" mentioned below in Section 4.8, it is not recommended except for machines with 64-bit words.) Computations involving complex quantities are always performed in complex arithmetic, and variables whose names begin with `C`, `W`, or `Z` are always complex. Each routine (other than those in the library routines described below) begins with the statements

```
IMPLICIT DOUBLE PRECISION (A-B,D-H,O-V,X-Y)
IMPLICIT COMPLEX*16(C,W,Z)
```

to make these type conventions automatic. On some machines the syntax of these statements may have to be modified.

3.3. History

My work on Schwarz-Christoffel mapping began in 1978 at the suggestion of Peter Henrici. SCPACK was developed originally during 1979–1980 on an IBM 370 installation at the Stanford Linear Accelerator Center, and then in 1983 on a VAX 750 and a Cyber 173 at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center. So far as I know, all versions of SCPACK currently in use date from the latter period and carry the label Version 2. Despite this Updated User's Guide, the SCPACK program is the same as in 1983.

Many others besides me have calculated numerical Schwarz-Christoffel maps over the years, with the most robust and efficient algorithms beginning to appear in 1979; particularly noteworthy has been the work of Reppe, Davis and Sridhar, Floryan and Zemach, Hoekstra, and Dias. References can be found in [2]. Some of these algorithms are more general than SCPACK, permitting curved boundary segments by way of the "continuous Schwarz-Christoffel formula." On the other hand some of them are less robust in the treatment of complicated regions and less efficient for high-accuracy calculations. So far as I know, SCPACK is the only publicly available software package for Schwarz-Christoffel mapping.

3.4. Routines included

SCPACK consists of twenty-one subroutines, four of which are most important for the user:

`QINIT`	computes Gauss-Jacobi quadrature nodes and weights
`SCSOLV`	solves the mapping problem for a given polygon
`WSC`	evaluates the forward map $w(z)$
`ZSC`	evaluates the inverse map $z(w)$

Depending on the application, the user may also wish to call:

`ANGLES`	computes angles, given vertices w_k
`RPROD`	computes $\|w'(z)\|^2$ at a point z
`NEARZ`	returns the number of the nearest prevertex z_k, etc.
`NEARW`	returns the number of the nearest vertex w_k, etc.
`ZQUAD`	computes the S-C integral between two points
`COUNT0`	initializes the log counter for machine-independent timing
`COUNT`	prints the current log count

Here are the other internal routines of SCPACK, to which the user will not normally need direct access:

`CHECK`	`SCFUN`	`SCTEST`	`ZQUAD1`	`ZQSUM`
`YZTRAN`	`SCOUTP`	`ZFODE`	`DIST`	`ZPROD`

In addition, three sets of library routines are required. `NS01A`, by M.J.D. Powell, solves a nonlinear system of equations whose solution is required for the Schwarz-Christoffel parameter problem. `NS01A` comes with six linear algebra routines from LINPACK for inverting a matrix. All together:

NS01A SGEFA SGEDI SSWAP
SAXPY SSCAL ISAMAX

QINIT requires GAUSSJ by G.H. Golub and J.H. Welsch, together with three subroutines from EISPACK, to compute nodes and weights for Gauss-Jacobi quadrature:

GAUSSJ CLASS IMTQL2* GAMMA*

Finally and least essentially, ZSC requires ODE and three associated subroutines, by Shampine and Gordon, to solve an ordinary differential equation:

ODE DE* STEP* INTRP

These codes can be deleted from the SCPACK collection (or replaced by dummies to prevent compiler complaints) if the user is prepared always to call ZSC with IGUESS = 1 (see Section 4.6).

The user is free to experiment with other comparable routines to replace these three library packages; SCPACK should still work satisfactorily after such substitutions. However, all of the routines listed above are included in the Netlib file sclibdbl and are known to have worked for thousands for conformal mapping problems. They are all in the public domain. (Subroutine GAMMA in the single precision version of SCPACK is an adaptation from the proprietary IMSL library, but it may be freely called as part of the GAUSSJ group provided it is not separated for general use.)

3.5. *Machine-dependent constants

The routines marked with asterisks above contain machine-dependent constants, as follows:

IMTQL2:	MACHEP = ϵ
DE:	FOURU = $4 * \epsilon$
STEP:	FOURU = $4 * \epsilon$, TWOU = $2 * \epsilon$
GAMMA:	various quantities

where ϵ is "machine epsilon", the smallest floating point number such that $1 + \epsilon > \epsilon$. In the standard version of SCPACK these numbers are set to generous values that should give satisfactory although not ideal performance on many machines.

Subroutine GAMMA (actually DGAMMA in the double precision version of SCPACK) contains many machine-dependent constants and should be replaced by a locally available gamma function routine if convenient.

3.6. Common blocks

SCPACK makes use of three labeled common blocks, /PARAM1/, /PARAM2/, and /LOGCNT/. The first two are needed for getting information to the functions SCFUN and ZFODE called by NS01A and ODE, respectively, whose calling sequences are fixed. The last saves logarithm counts for the purpose of machine-independent timing by subroutines COUNT0 and COUNT (see Section 4.7).

4. Use of SCPACK

4.1. Outline

To compute the Schwarz-Christoffel transformation from the unit disk to a given polygon, here is what to do:

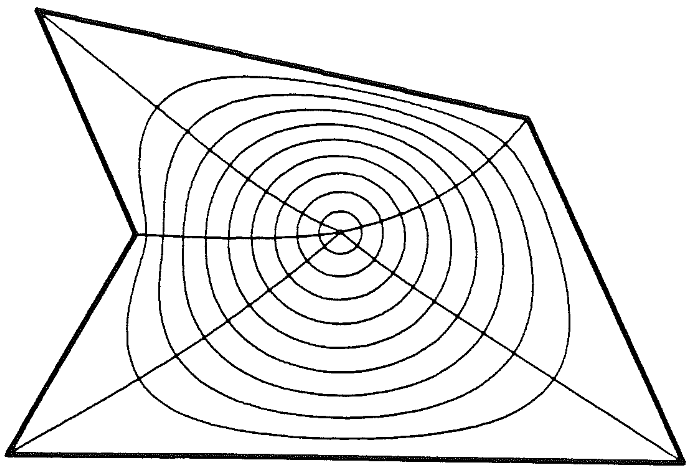
1) Set `N` (N, the number of vertices), `W(K)` (w_k, the vertices), `NPTSQ` (the number of quadrature points per subinterval), and `BETAM(K)` ($-\beta_k$, the exterior turning angle at w_k divided by $-\pi$; see the figure on p. 3). Each parameter `BETAM(K)` can be computed by subroutine `ANGLES` if the vertices `W(K-1)`, `W(K)`, and `W(K+1)` are all finite.

2) Call `QINIT` to compute nodes and weights for Gauss-Jacobi quadrature.

3) Set `WC` ($w_c = w(0)$, a point interior to the polygon) and the other input parameters to `SCSOLV`, and call `SCSOLV` to solve the parameter problem for the constant `C` (C) and the prevertices `Z(K)` (z_k).

4) Map individual points as desired from the disk to the polygon with routine `WSC` and from the polygon to the disk with routine `ZSC`.

More complicated problems may require other sequences of computations. For example, two Schwarz-Christoffel maps can be composed by calling `QINIT` and `SCSOLV` twice, using distinct variables in the calling sequences. This is necessary whenever one polygon is mapped onto another with the disk as an intermediate domain. See Section 5.3 below for an example.



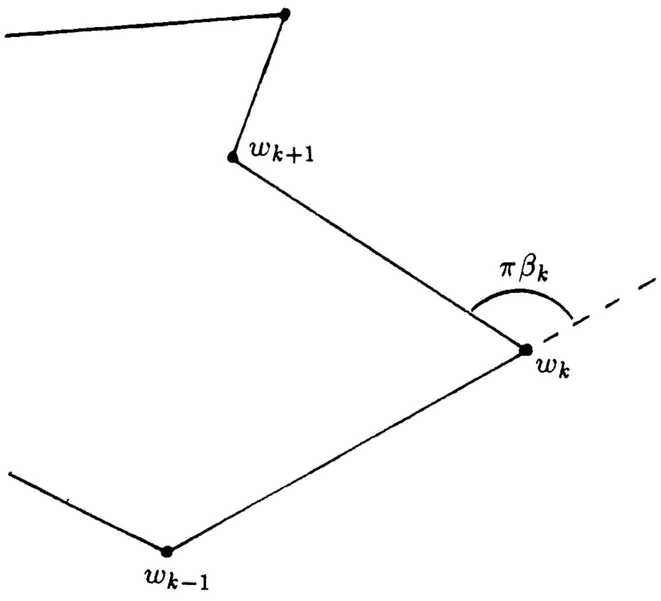
4.2. ANGLES: compute exterior turning angles

Most SCPACK routines require as input the vector `BETAM(1)`,...,`BETAM(N)` of angle parameters. Each value `BETAM(K)` is equal to the exterior turning angle in the polygon at vertex `W(K)` divided by $-\pi$; see the figure on p. 3. The subroutine `ANGLES` computes these angle parameters from the vertices. Here is the calling sequence:

▶ `SUBROUTINE ANGLES(N,W,BETAM)`

`N`	Number of vertices of the polygon *(input)*. Must be ≤ 20. See additional comments in the documentation for `SCSOLV`.
`W`	Complex array of vertices `W(1)`,...,`W(N)` of the polygon *(input)*. See additional comments in the documentation for `SCSOLV`.
`BETAM`	Real array of angle parameters *(output)*. See additional comments in the documentation for `SCSOLV`.

If all N vertices `W(K)` are finite, `ANGLES` will compute N values `BETAM(K)`. If some of them are infinite, then `BETAM(K)` is meaningful only for those vertices `W(K)` such that `W(K-1)`, `W(K)`, and `W(K+1)` are all finite. See the listing of `ANGLES` on p. 36 for more details.



4.3. QINIT: initialization for Gauss-Jacobi quadrature

QINIT must be called before SCSOLV, WSC, or ZSC. It computes the Gauss-Jacobi quadrature nodes and weights that make integration of the Schwarz-Christoffel formula possible.

▶ `SUBROUTINE QINIT(N,BETAM,NPTSQ,QWORK)`

N	Number of vertices of the polygon *(input)*. Must be ≤ 20. See additional comments in the documentation for SCSOLV.
BETAM	Real array of angle parameters *(input)*. See additional comments in the documentation for SCSOLV.
NPTSQ	Number of points per subinterval in Gauss-Jacobi quadrature *(input)*. Recommended value: equal to the number of digits of accuracy desired in the answer, e.g. NPTSQ = 6 if you want about 6-digit accuracy. Must be ≥ 2. Total SCPACK computation time increases linearly with NPTSQ.
QWORK	Real work array dimensioned at least NPTSQ*(2N+3) *(output)*. On output, QWORK will contain quadrature nodes and weights corresponding to the vertices of the polygon; SCPACK users can ignore the details. The same array with the same value NPTSQ must be given later as input to SCSOLV, WSC, or ZSC. If more than one Schwarz-Christoffel map is to be composed, then more than one copy of QWORK must be filled by QINIT and kept for SCSOLV, WSC, or ZSC.

4.4. SCSOLV: solution of the parameter problem

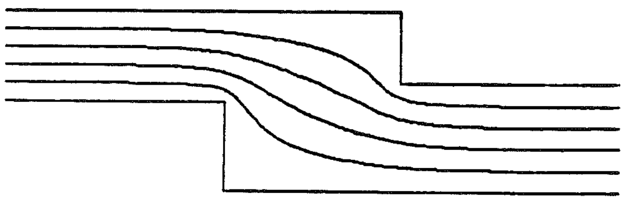
SCSOLV solves the parameter problem for the constant C and the prevertices Z(K) of the Schwarz-Christoffel map (∗). If the problem involves a map of the disk onto a prescribed polygon, SCSOLV must be called before this map is evaluated with WSC or ZSC. In rarer applications where the prevertices Z(K) are prescribed rather than the vertices W(K), SCSOLV is not needed. This occurs below in the example of Section 5.3.

► SUBROUTINE SCSOLV(IPRINT,IGUESS,TOL,ERREST,N,C,Z,WC,W,BETAM,NPTSQ,QWORK)

IPRINT	-2, -1, 0, or 1 for increasing amounts of diagnostic output *(input)*.
IGUESS	1 if an initial guess for Z is being supplied, otherwise 0 *(input)*.
TOL	Desired accuracy in solution of nonlinear system *(input)*. Recommended value: $10^{-(\mathrm{NPTSQ}+1)}\times$ typical size of vertices W(K).
ERREST	Estimated error in solution *(output)*.
N	Number of vertices of the polygon *(input)*. Must be ≤ 20.
C	Complex scale factor in formula (∗) *(output)*.
Z	Complex array of prevertices Z(1),...,Z(N) on the unit circle *(output)*. Dimension at least N. If an initial guess is being supplied it should be in Z on input, with Z(N) $=1$. In any case the correct prevertices will be in Z on output.
WC	Complex image of 0 in the polygon, as in formula (∗) *(input)*. It is safest to pick WC to be as central as possible in the polygon in the sense that as little of the polygon as possible is shielded from WC by reentrant edges.
W	Complex array of vertices W(1),...,W(N) of the polygon *(input)*. Dimension at least N. It is a good idea to keep W(K) roughly on the scale of unity. W(K) will be ignored when the vertex lies at infinity as defined by BETAM, below. Each connected boundary component must include at least one vertex W(K), even if it has to be a degenerate vertex with BETAM(K) = 0. W(N) and W(1) must be finite.
BETAM	Real array of angle parameters *(input)*. Each BETAM(K) is equal to the exterior turning angle in the polygon at vertex W(K) divided by $-\pi$ (see the figure on p. 3). Dimension at least N. Permitted values lie in the range $-3\leq$ BETAM(K) ≤ 1. Examples: BETAM(K) is -1 for a parallel channel extending to ∞, $-1/2$ at each corner of a rectangle, $-2/3$ at each corner of an equilateral triangle, $+1$ at the end of a slit. The sum of the numbers BETAM(K) will be -2 if they have been set correctly, and SCPACK will complain if the sum differs from -2 by more than TOL. BETAM(N-1) may not be 0 or 1. W(K) is assumed to lie at infinity if and only if BETAM(K) ≤ -1.

NPTSQ Number of points per subinterval in Gauss-Jacobi quadrature *(input)*. Recommended value: equal to the number of digits of accuracy desired in the answer, e.g. NPTSQ = 6 if you want about 6-digit accuracy. Must be ≥ 2, and must be the same as in the call to QINIT which filled the vector QWORK.

QWORK Real quadrature work array *(input)*. Dimension at least NPTSQ$\times$(2N+3) but no greater than 460. Before calling SCSOLV, QWORK must have been filled by subroutine QINIT.



4.5. WSC and NEARZ: evaluation of the forward map

Once the parameter problem has been solved, the function WSC is available to evaluate the forward map: $w = w(z)$. WSC is a complex function with calling sequence

► FUNCTION WSC(ZZ,KZZ,ZO,WO,KO,N,C,Z,BETAM,NPTSQ,QWORK)

The value returned will be the image of ZZ in the polygon.

ZZ	Point in the disk at which W(ZZ) is desired *(input).*
KZZ	=K if ZZ = Z(K) for some K, otherwise 0 *(input).*
ZO	Nearby point in the disk at which W(ZO) is known and finite *(input).*
WO	W(ZO) *(input).*
KO	=K if ZO = Z(K) for some K, otherwise 0 *(input).*

N,C,Z,BETAM,NPTSQ,QWORK. As in SCSOLV *(input).*

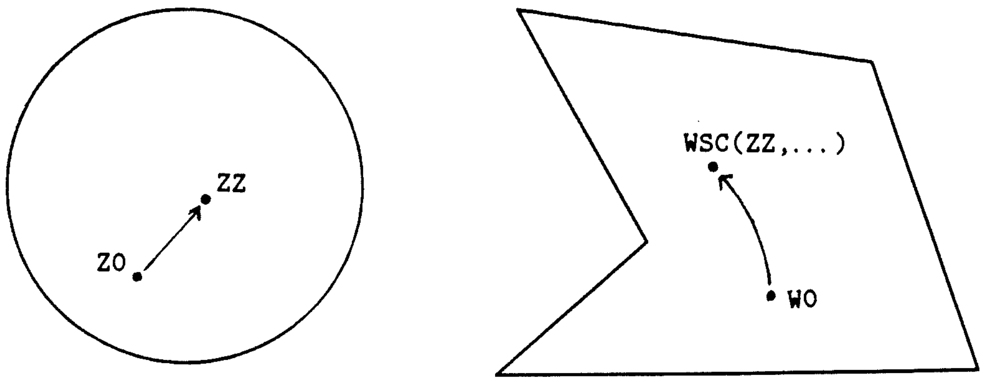
ZO should not be too far from ZZ. A simple and adequate choice is the nearest prevertex Z(K) with W(K) finite, or 0 if 0 is closer than any Z(K). This choice of ZO can be determined automatically by the function NEARZ, with the following calling sequence:

► SUBROUTINE NEARZ(ZZ,ZO,WO,KO,N,Z,WC,W,BETAM)

Thus a common way to compute W(ZZ) for an arbitrary value ZZ is with the following sequence of commands:

```
CALL NEARZ(ZZ,ZO,WO,KO,N,Z,WC,W,BETAM)
WW = WSC(ZZ,0,ZO,WO,KO,N,C,Z,BETAM,NPTSQ,QWORK)
```

See Section 5.1 for an example. However, sometimes this pair of commands is not the most efficient way to proceed — for example, if you are calculating successive points along a curve, where a good choice of ZO is simply the previous point on the curve — and that is why NEARZ and WSC are separate programs. For further information about NEARZ, see its listing on p. 35.



4.6. ZSC and NEARW: evaluation of the inverse map

ZSC is analogous to WSC but evaluates the inverse map $z = z(w)$:

▶ FUNCTION ZSC(WW,IGUESS,ZINIT,ZO,WO,KO,EPS,IER,N,C,Z,WC,W,BETAM,NPTSQ,QWORK)

The value returned will be the preimage of WW in the disk.

WW	Point in the polygon at which Z(WW) is desired *(input).*
IGUESS	*(input).* =1: Initial guess is supplied as parameter ZINIT. ≠1: Get initial guess from program ODE (slower). For this the line segment from WC to WW must lie within the polygon.
ZINIT	Initial guess if IGUESS=1, otherwise ignored *(input).* May not be a prevertex Z(K).
ZO	Point in the disk near Z(WW) at which W(ZO) is known and finite *(input).*
WO	W(ZO) *(input).* The line segment from WO to WW must lie in the interior of the polygon.
KO	= K if ZO = Z(K) for some K, otherwise 0 *(input).*
EPS	Desired accuracy in answer Z(WW) *(input).*
IER	Error flag *(input and output).* On input, give IER ≠ 0 to suppress error messages. On output, IER ≠ 0 indicates unsuccessful computation — try again with a better initial guess.

N,C,Z,WC,W,BETAM,NPTSQ,QWORK. As in SCSOLV *(input).*

As with WSC, here WO should not be too far from WW. In analogy to NEARZ, the routine NEARW selects WC or the nearest vertex W(K) for WO, a choice that should be suitable for many applications:

▶ SUBROUTINE NEARW(WW,ZO,WO,KO,N,Z,WC,W,BETAM)

Thus a common way to compute Z(WW) for some specified value WZ is with the following seqquence of commands:

```
CALL NEARW(WW,ZO,WO,KO,N,Z,WC,W,BETAM)
IER = 0
ZZ = ZSC(WW,0,ZO,ZO,WO,KO,TOL,IER,N,C,Z,WC,W,BETAM,NPTSQ,QWORK)
IF (IER.NE.0) PRINT *, 'ERROR IN ZSC'
```

For further information about NEARW, see its listing on p. 36.

Note that the segment WO–WW must lie in the interior of the polygon. In treating non-convex polygons the user must bear this in mind when calling ZSC — it may occasionally be necessary first to find the preimage of an intermediate point WO′.

4.7. COUNT0 and COUNT

As mentioned in Section 3.1, the bottleneck in SCPACK computations is the calculation of a complex logarithm in subroutine ZPROD (p. 34 of this User's Guide). A significant proportion of the computer time in an SCPACK run may be spent simply evaluating this logarithm. This unfortunate fact has a fortunate consequence: by counting complex logarithms, we can obtain a rough estimate of execution time that is machine-independent. COUNT0 and COUNT are designed to do the counting.

► `SUBROUTINE COUNT0`

► `SUBROUTINE COUNT`

These subroutines take no arguments. A call to COUNT0 (re)sets the counter to zero. Subsequent calls to COUNT generate a report of the number of logarithms since the last call and in total. The output might look like this:

```
------- NO. LOGS: SINCE LAST COUNT    32442,    TOTAL   862219
```

An example appears in Section 5.3. It is also easy to access these numbers for uses other than printing; see the listings on p. 37.

4.8. Limitations, bugs, and remarks

I do not know of any bugs in SCPACK, though undoubtedly there are some. However, here are some features to watch out for:

The "crowding" phenomenon. Beware of elongated polygons! For well-understood reasons related to the exponential decay of boundary information in analytic functions, they can be very difficult to treat numerically. For example, SCPACK will probably fail if you ask it to map a rectangle of aspect ratio 20. This limitation has proved to be the principal shortcoming of SCPACK in practical applications. To get around the problem one normally has to subdivide the polygon or dispense with the use of a disk (or half-plane) as a fundamental domain. Polygons that are highly elongated in one direction only can be effectively mapped by a Schwarz-Christoffel formula based on an infinite strip [9], but SCPACK does not provide this option; contact me or Louis H. Howell for information about an experimental computer program.

$N \le 20$. As mentioned above, the number of vertices N must be ≤ 20. This is an artificial restriction imposed because NS01A happens to be dimensioned that way. The user who wants to treat larger polygons can do so by modifying a few `DIMENSION` statements.

Disk vs. half-plane. Despite a remark to the contrary in [1], SCPACK could equally well have been designed to map a half-plane rather than a disk. The disk has the advantages of symmetry and finiteness of all numbers. The half-plane has the advantage that most of the integrals involved in solving the parameter problem can be reduced to real variables with a resulting speed-up of a factor of 2 or so.

User-specified point $w_c = w(0)$. SCPACK squanders another factor of 2 or so by fixing one interior point ($w_c = w(0)$) and one boundary point (`Z(N)` $= 1$) rather than three boundary points. As a result the parameter problem involves $N-1$ unknowns rather than $N-3$ — a significant difference since this number gets cubed. If I were writing SCPACK again now I would do it differently.

Rectangles and elliptic functions. Maps of a polygon to a rectangle come up frequently in applications, and can be treated by composing two SCPACK maps. One of these is a Jacobian elliptic function, and could be treated very quickly by special methods based on the arithmetic-geometric mean iteration (see the Appendix to [9]). However, since the other half of the composition is usually more time-consuming anyway, I have not considered it worthwhile to include such an option in SCPACK.

Adaptive contour plotting. Plotting contours (e.g. equipotentials or streamlines) is often more time-consuming than solving the parameter problem. Here are three tips to speed up the process: 1) set `NPTSQ` $= 2$ or 3, even if a higher value was used in solving the parameter problem; 2) calculate each new point along a curve by integration with `WSC` or `ZQUAD` along the short line segment from the previous point; 3) most important, write yourself a curve plotting routine that spaces points adaptively according to the local curvature.

The inverse map `ZSC`. This is the least carefully designed part of SCPACK, and the most likely to fail to perform as a reliable black box.

5. Examples

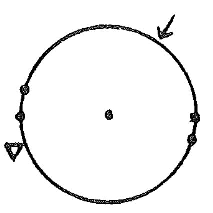
5.1. EX1: map a few points back and forth

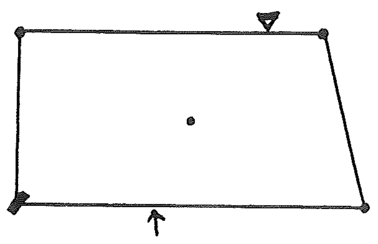
Our first example maps the disk onto a finite polygon input online. After solving the parameter problem, the program enters an infinite loop in which it alternately computes $w(z)$ and $z(w(z))$ for specified z, then $z(w)$ and $w(z(w))$ for specified w. Of course, such an example is more fun if you can input points with a mouse.

```
      PROGRAM EX1
      IMPLICIT REAL*8(A-B,D-H,O-V,X-Y), COMPLEX*16(C,W,Z)
      DIMENSION Z(20),W(20),BETAM(20),QWORK(460)
C
C SPECIFY GEOMETRY:
      PRINT *, 'ALL COMPLEX NUMBERS SHOULD BE INPUT AS PAIRS (X,Y)'
      PRINT *, 'NUMBER OF VERTICES?'
       READ *, N
      PRINT *, 'VERTICES?'
       READ *, (W(K),K=1,N)
      PRINT *, 'CENTRAL POINT WC = W(0)?'
       READ *, WC
      PRINT *, 'NUMBER OF DIGITS DESIRED?'
       READ *, NDIG
C
C SOLVE PARAMETER PROBLEM:
      NPTSQ = NDIG
      TOL = 10.**(-NDIG-1)
      CALL ANGLES(N,W,BETAM)
      CALL QINIT(N,BETAM,NPTSQ,QWORK)
      CALL SCSOLV(0,0,TOL,ERREST,N,C,Z,WC,W,BETAM,NPTSQ,QWORK)
C
C MAP A POINT FORWARD AND THEN BACK:
    1 PRINT *, 'POINT ZZ TO BE MAPPED?'
      READ *, ZZ
      CALL NEARZ(ZZ,Z0,W0,K0,N,Z,WC,W,BETAM)
      WW = WSC(ZZ,0,Z0,W0,K0,N,C,Z,BETAM,NPTSQ,QWORK)
      CALL NEARW(WW,Z0,W0,K0,N,Z,WC,W,BETAM)
      IER = 0
      ZZZ = ZSC(WW,0,Z0,Z0,W0,K0,TOL,IER,N,C,Z,WC,W,BETAM,NPTSQ,QWORK)
      IF (IER.NE.0) PRINT *, 'ERROR IN ZSC'
      PRINT *, '        W(ZZ)  =', WW
      PRINT *, '     Z(W(ZZ)) =', ZZZ
C
C MAP A POINT BACK AND THEN FORWARD:
      PRINT *, 'POINT WW TO BE INVERSE MAPPED?'
      READ *, WW
      CALL NEARW(WW,Z0,W0,K0,N,Z,WC,W,BETAM)
      IER = 0
      ZZ = ZSC(WW,0,Z0,Z0,W0,K0,TOL,IER,N,C,Z,WC,W,BETAM,NPTSQ,QWORK)
      IF (IER.NE.0) PRINT *, 'ERROR IN ZSC'
      CALL NEARZ(ZZ,Z0,W0,K0,N,Z,WC,W,BETAM)
      WWW = WSC(ZZ,0,Z0,W0,K0,N,C,Z,BETAM,NPTSQ,QWORK)
      PRINT *, '        Z(WW)  =', ZZ
      PRINT *, '     W(Z(WW)) =', WWW
      GOTO 1
      END
```

Here is a sample run:

```
 ALL COMPLEX NUMBERS SHOULD BE INPUT AS PAIRS (X,Y)
 NUMBER OF VERTICES?
 4
 VERTICES?
 (2,-1)
 (1.5,1)
 (-2,1)
 (-2,-1)
 CENTRAL POINT WC = W(0)?
 (0,0)
 NUMBER OF DIGITS DESIRED?
 6
    THE SUM-OF-SQUARES ERROR AT STEP   1 IS 0.3141E+01
    THE SUM-OF-SQUARES ERROR AT STEP   2 IS 0.3141E+01
    THE SUM-OF-SQUARES ERROR AT STEP   3 IS 0.3141E+01
    THE SUM-OF-SQUARES ERROR AT STEP   4 IS 0.3141E+01
    THE SUM-OF-SQUARES ERROR AT STEP   5 IS 0.6309E+00
    THE SUM-OF-SQUARES ERROR AT STEP   6 IS 0.4330E+00
    THE SUM-OF-SQUARES ERROR AT STEP   7 IS 0.1165E+00
    THE SUM-OF-SQUARES ERROR AT STEP   8 IS 0.6782E-01
    THE SUM-OF-SQUARES ERROR AT STEP   9 IS 0.1643E-02
    THE SUM-OF-SQUARES ERROR AT STEP  10 IS 0.5306E-03
    THE SUM-OF-SQUARES ERROR AT STEP  11 IS 0.7583E-04
    THE SUM-OF-SQUARES ERROR AT STEP  12 IS 0.2125E-05
    THE SUM-OF-SQUARES ERROR AT STEP  13 IS 0.1037E-09
    THE SUM-OF-SQUARES ERROR AT STEP  14 IS 0.5897E-12
    THE SUM-OF-SQUARES ERROR AT STEP  15 IS 0.1217E-13
    THE SUM-OF-SQUARES ERROR AT STEP  16 IS 0.1074E-11
    THE SUM-OF-SQUARES ERROR AT STEP  17 IS 0.1105E-15
```



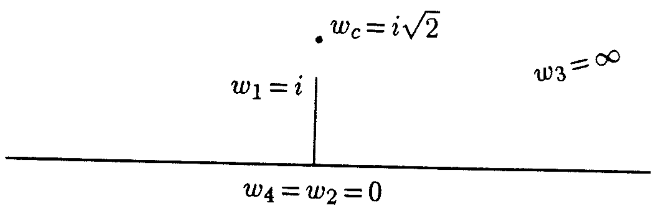


```
PARAMETERS DEFINING MAP:                  (N =  4)      (NPTSQ =  6)

 K           W(K)             TH(K)/PI     BETAM(K)            Z(K)
---          ----             --------     --------            ----

 1     ( 2.000,-1.000)      0.87295887    -0.57798    (-0.92140664, 0.38859980)
 2     ( 1.500, 1.000)      1.04402175    -0.42202    (-0.99045201,-0.13785798)
 3     (-2.000, 1.000)      1.89051229    -0.50000    ( 0.94142472,-0.33722323)
 4     (-2.000,-1.000)      2.00000000    -0.50000    ( 1.00000000, 0.00000000)

WC = ( 0.00000000E+00, 0.00000000E+00)
 C = (-0.12387514E+01,-0.21538050E+00)

ERREST:  0.1535E-07

POINT ZZ TO BE MAPPED?
(.6,.8)
        W(ZZ)  =  (  -0.31247286254467,  -0.99999999683141)
      Z(W(ZZ)) =  (   0.60000000997326,   0.79999999911476)
POINT WW TO BE INVERSE MAPPED?
(1,1)
        Z(WW)  =  (  -0.95991116379178,  -0.28030440347903)
      W(Z(WW)) =  (   1.00000000000000,   1.00000000000000)
POINT ZZ TO BE MAPPED?
```

5.2. EX2: half-plane with slit

The next example program computes a Schwarz-Christoffel map that is known analytically: onto a half-plane with a slit. A few calculated values for the forward map WSC are then compared with the exact answers. This is an example of a polygon with a vertex at infinity.

```
      PROGRAM EX2
      IMPLICIT REAL*8(A-B,D-H,O-Y), COMPLEX*16(C,W,Z)
      DIMENSION Z(20),W(20),BETAM(20),QWORK(344)
      ZERO = (0.D0,0.D0)
      ZI = (0.D0,1.D0)
C
C SPECIFY GEOMETRY:
      N = 4
      WC = ZI * SQRT(2.D0)
      W(1) = ZI
      W(2) = ZERO
      W(3) = (1.D20,1.D20)
      W(4) = ZERO
      BETAM(1) =  1.0
      BETAM(2) = -0.5
      BETAM(3) = -2.0
      BETAM(4) = -0.5
C
C SOLVE PARAMETER PROBLEM:
C   (INITIAL GUESS IS PROVIDED TO PREVENT ACCIDENTAL EXACT SOLUTION)
      NPTSQ = 8
      CALL QINIT(N,BETAM,NPTSQ,QWORK)
      IPRINT = 0
      IGUESS = 1
      DO 1 K = 1,4
    1   Z(K) = EXP(ZI*(K-4))
      TOL = 1.D-9
      CALL SCSOLV(IPRINT,IGUESS,TOL,ERREST,N,C,Z,
     &  WC,W,BETAM,NPTSQ,QWORK)
C
C COMPARE WSC(Z) TO EXACT VALUES FOR VARIOUS Z:
      DO 10 I = 1,4
        ZZ = .3D0 * DCMPLX(I-2.D0,.2D0*I+.5D0)
        WW = WSC(ZZ,0,ZERO,WC,0,N,C,Z,BETAM,NPTSQ,QWORK)
        ZTMP = -ZI * (ZZ-ZI) / (ZZ+ZI)
        WWEX = ZI * SQRT(-ZTMP**2 + 1.D0)
        ERR = ABS(WW-WWEX)
        WRITE (6,201) ZZ,WW,WWEX,ERR
   10   CONTINUE
C
  201 FORMAT (' Z,W,WEX,ERR: ',3('(',F6.3,',',F6.3,') '),D11.4)
      END
```



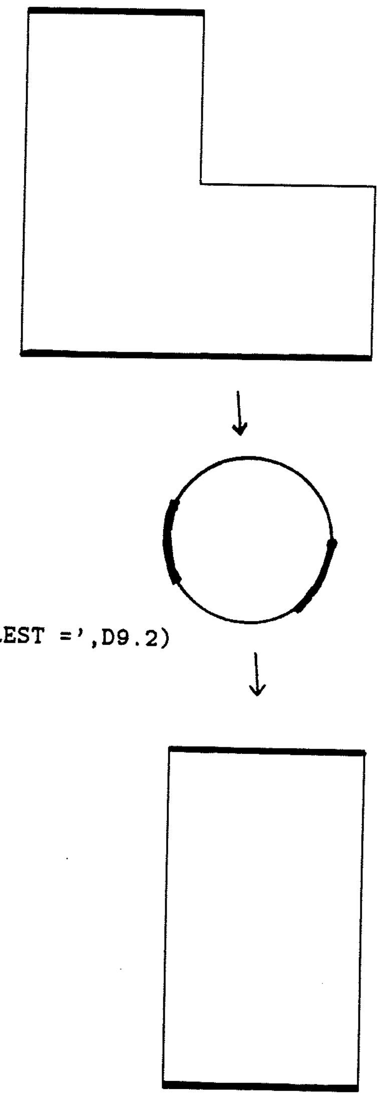
A Sun 3/50 Workstation produced the following output in about 6 seconds of elapsed time:

```
   THE SUM-OF-SQUARES ERROR AT STEP   1 IS 0.4628E+00
   THE SUM-OF-SQUARES ERROR AT STEP   2 IS 0.4628E+00
   THE SUM-OF-SQUARES ERROR AT STEP   3 IS 0.4628E+00
   THE SUM-OF-SQUARES ERROR AT STEP   4 IS 0.4628E+00
   THE SUM-OF-SQUARES ERROR AT STEP   5 IS 0.3426E+00
   THE SUM-OF-SQUARES ERROR AT STEP   6 IS 0.2393E-01
   THE SUM-OF-SQUARES ERROR AT STEP   7 IS 0.1367E+00
   THE SUM-OF-SQUARES ERROR AT STEP   8 IS 0.1084E-02
   THE SUM-OF-SQUARES ERROR AT STEP   9 IS 0.1240E-04
   THE SUM-OF-SQUARES ERROR AT STEP  10 IS 0.2607E-05
   THE SUM-OF-SQUARES ERROR AT STEP  11 IS 0.1611E-06
   THE SUM-OF-SQUARES ERROR AT STEP  12 IS 0.1487E-09
   THE SUM-OF-SQUARES ERROR AT STEP  13 IS 0.3594E-12
   THE SUM-OF-SQUARES ERROR AT STEP  14 IS 0.1277E-16
   THE SUM-OF-SQUARES ERROR AT STEP  15 IS 0.3803E-21

  PARAMETERS DEFINING MAP:                (N =  4)      (NPTSQ =  8)

  K          W(K)           TH(K)/PI     BETAM(K)            Z(K)
 ---         ----           --------     --------            ----

  1     ( 0.000, 1.000)    0.50000000     1.00000    ( 0.00000000, 1.00000000)
  2     ( 0.000, 0.000)    1.00000000    -0.50000    (-1.00000000, 0.00000000)
  3         INFINITY       1.50000000    -2.00000    ( 0.00000000,-1.00000000)
  4     ( 0.000, 0.000)    2.00000000    -0.50000    ( 1.00000000, 0.00000000)

 WC = ( 0.00000000E+00, 0.14142136E+01)
  C = (-0.14142136E+01,-0.18626164E-10)

 ERREST:  0.1806E-10

 Z,W,WEX,ERR: (-0.300, 0.210) ( 0.196, 1.095) ( 0.196, 1.095)  0.1843D-10
 Z,W,WEX,ERR: ( 0.000, 0.270) ( 0.000, 1.153) ( 0.000, 1.153)  0.9080D-11
 Z,W,WEX,ERR: ( 0.300, 0.330) (-0.133, 1.048) (-0.133, 1.048)  0.9255D-11
 Z,W,WEX,ERR: ( 0.600, 0.390) (-0.126, 0.887) (-0.126, 0.887)  0.9350D-11
```

5.3. EX3: call RESIST to compute a resistance

Our final example illustrates how two Schwarz-Christoffel maps can be composed, and it also illustrates the use of COUNTO and COUNT. The problem is to compute the resistance (conformal modulus) of an L-shaped hexagon, assuming that a voltage difference is applied between the top and the bottom edges. In other words the question is, what is the length-to-width ratio of a rectangle which is conformally equivalent to the L-shaped region, provided that the four distinguished vertices map to the corners? The exact answer is $\sqrt{3} \approx 1.732050807569$. The basis of the computation is a subroutine RESIST, a driver for SCPACK that composes two Schwarz-Christoffel maps in such a way as to map a polygon conformally onto a rectangle and thereby determine the dimensions of a rectangle which is electrically equivalent to a given polygonal resistor. RESIST is listed in Section 7.2, and to understand this example, please take a look at it.

```
      PROGRAM EX3
      IMPLICIT REAL*8(A-B,D-H,O-Y), COMPLEX*16(C,W,Z)
      DIMENSION W(10),IBRK(4),QWORK(300)
      N = 6
      W(1) = (0.D0,0.D0)
      W(2) = (2.D0,0.D0)
      W(3) = (2.D0,1.D0)
      W(4) = (1.D0,1.D0)
      W(5) = (1.D0,2.D0)
      W(6) = (0.D0,2.D0)
      WC = (.5D0,.5D0)
      IBRK(1) = 1
      IBRK(2) = 2
      IBRK(3) = 5
      IBRK(4) = 6
C
C MAIN LOOP: DIFFERENT ACCURACY SPECIFICATIONS:
      CALL COUNTO
      DO 10 NDIG = 2,10,2
        R = RESIST(N,W,WC,IBRK,NDIG,ERREST,QWORK)
   10   WRITE (6,201) NDIG,R,ERREST
      CALL COUNT
C
  201 FORMAT ('    NDIG =',I3,':','    R =',D20.13,',    ERREST =',D9.2)
      END
```



After about 80 seconds on the Sun 3/50, here is the output:

```
------- LOG COUNTER SET TO ZERO
NDIG =  2:    R = 0.1732597161966D+01,   ERREST = 0.12D-02
NDIG =  4:    R = 0.1732040998524D+01,   ERREST = 0.15D-04
NDIG =  6:    R = 0.1732050829623D+01,   ERREST = 0.55D-07
NDIG =  8:    R = 0.1732050807808D+01,   ERREST = 0.76D-09
NDIG = 10:    R = 0.1732050807573D+01,   ERREST = 0.23D-10
------- NO. LOGS: SINCE LAST COUNT  75880,    TOTAL   75880
```

6. References

As mentioned in the Introduction, the design of SCPACK is described in

[1] L.N. Trefethen, "Numerical computation of the Schwarz-Christoffel transformation," *SIAM J. Sci. Stat. Comput. 1* (1980), 82–102,

and a survey of various applications of Schwarz-Christoffel mapping, and of variations on the Schwarz-Christoffel theme, can be found in

[2] L.N. Trefethen, "Schwarz-Christoffel mapping in the 1980's," Numerical Analysis Report 89-1, Dept. of Mathematics, M.I.T., 1989 (available from L.N.T.).

Here again is the Netlib reference:

[3] J. J. Dongarra and E. Grosse, "Distribution of mathematical software via electronic mail," *Communications of the ACM 30* (1987), 403–407.

The report [2] contains many references, including a fairly complete list of other authors who have published on the subject of numerical Schwarz-Christoffel mapping. Among the Schwarz-Christoffel variations it discusses are the treatment of exterior polygons, circular arc polygons, periodic polygons, polygonal Riemann surfaces, doubly-connected polygons, and "gearlike domains". The extension to doubly-connected polygons is particularly noteworthy:

[4] H. Däppen, *Die Schwarz-Christoffel-Abbildung für zweifach zusammenhängende Gebiete mit Anwendungen,* PhD dissertation, Dept. of Mathematics, ETH-Zurich, 1988.

For numerical conformal mapping in general (mainly integral equation methods for domains with curved boundaries) there are three references in book form,

[5] D. Gaier, *Konstruktive methoden der Konformen Abbildung,* Springer, 1964,

[6] P. Henrici, *Applied and Computational Complex Analysis, v. 3,* Wiley, 1986,

[7] L. N. Trefethen, ed., *Numerical Conformal Mapping,* North-Holland, 1986,

and various survey articles, such as

[8] D. C. Ives, "Conformal grid generation," in J. F. Thompson, ed., *Numerical Grid Generation,* Elsevier, 1982.

The survey article by Gutknecht in [7] should also be mentioned. The extension of Schwarz-Christoffel mapping for highly-elongated polygons mentioned in Section 4.8 — to avoid the problem of "crowding" — is described in

[9] L. H. Howell and L. N. Trefethen, "A modified Schwarz-Christoffel transformation for elongated regions," *SIAM J. Sci. Stat. Comput.,* to appear.

7. Program listing (excluding library routines)

7.1. SCPACK

```
CC      *** SCPACK VERSION 2 ***          LLOYD NICHOLAS TREFETHEN
CC
CC      HISTORY:
CC        SCPACK 1 - OCT. 1979
CC        SCPACK 2 - JULY 1983
CC
CC      SCPACK IS A FORTRAN PACKAGE FOR THE NUMERICAL IMPLEMENTATION
CC      OF THE SCHWARZ-CHRISTOFFEL CONFORMAL MAP FROM A DISK TO
CC      A POLYGON, WHICH MAY CONTAIN VERTICES AT INFINITY.  SEE
CC      TREFETHEN, "NUMERICAL COMPUTATION OF THE SCHWARZ-CHRISTOFFEL
CC      TRANSFORMATION", SIAM J. SCI. STAT. COMP. 1 (1980), 82-102.
CC      THE PACKAGE MAY BE FREELY COPIED AND USED, BUT REFERENCE TO
CC      THE ABOVE PAPER SHOULD BE GIVEN IN ANY PUBLICATIONS ARISING
CC      FROM ITS USE.
CC
CC      COMMENT CARDS DESCRIBE THE PRINCIPAL FEATURES OF THE PACKAGE.
CC      ADDITIONAL DOCUMENTATION IS AVAILABLE IN THE "SCPACK USER'S
CC      GUIDE".
CC
CC      THE PACKAGE REQUIRES LIBRARY ROUTINES NS01A, GAUSSJ, AND ODE,
CC      AND THESE CONTAIN MACHINE DEPENDENT CONSTANTS.  SEE THE SCPACK
CC      USER'S GUIDE.


C*********************************************************************
C* QINIT                                      PRIMARY SUBROUTINE  **
C*********************************************************************
      SUBROUTINE QINIT(N,BETAM,NPTSQ,QWORK)
C
C COMPUTES NODES AND WEIGHTS FOR GAUSS-JACOBI QUADRATURE
C
C CALLING SEQUENCE PARAMETERS: SEE COMMENTS IN SCSOLV
C
C THE WORK ARRAY QWORK MUST BE DIMENSIONED AT LEAST NPTSQ * (2N+3).
C IT IS DIVIDED UP INTO 2N+3 VECTORS OF LENGTH NPTSQ: THE FIRST
C N+1 CONTAIN QUADRATURE NODES ON OUTPUT, THE NEXT N+1 CONTAIN
C QUADRATURE WEIGHTS ON OUTPUT, AND THE FINAL ONE IS A
C SCRATCH VECTOR NEEDED BY GAUSSJ.
C
      IMPLICIT DOUBLE PRECISION (A-B,D-H,O-V,X-Y)
      IMPLICIT COMPLEX*16(C,W,Z)
      DIMENSION QWORK(1),BETAM(N)
C
C FOR EACH FINITE VERTEX W(K), COMPUTE NODES AND WEIGHTS FOR
C ONE-SIDED GAUSS-JACOBI QUADRATURE ALONG A CURVE BEGINNING AT Z(K):
      ISCR = NPTSQ*(2*N+2) + 1
      DO 1 K = 1,N
        INODES = NPTSQ*(K-1) + 1
        IWTS = NPTSQ*(N+K) + 1
    1   IF (BETAM(K).GT.-1.D0) CALL GAUSSJ(NPTSQ,0.D0,BETAM(K),
     &     QWORK(ISCR),QWORK(INODES),QWORK(IWTS))
C
C COMPUTE NODES AND WEIGHTS FOR PURE GAUSSIAN QUADRATURE:
      INODES = NPTSQ*N + 1
      IWTS = NPTSQ*(2*N+1) + 1
      CALL GAUSSJ(NPTSQ,0.D0,0.D0,
     &  QWORK(ISCR),QWORK(INODES),QWORK(IWTS))
C
      RETURN
      END
```

```
C*****************************************************************************
C* SCSOLV                                                 PRIMARY SUBROUTINE **
C*****************************************************************************
C
      SUBROUTINE SCSOLV(IPRINT,IGUESS,TOL,ERREST,N,C,Z,WC,
     &   W,BETAM,NPTSQ,QWORK)
C
C THIS SUBROUTINE COMPUTES THE ACCESSORY PARAMETERS C AND
C Z(K) FOR THE SCHWARZ-CHRISTOFFEL TRANSFORMATION
C WHICH SENDS THE UNIT DISK TO THE INTERIOR OF THE POLYGON
C W(1),...,W(N).  THIS MAPPING IS OF THE FORM:
C
C                           Z    N
C    W   =   WC  +   C * INT PROD (1-Z/Z(K))**BETAM(K) DZ .     (1.2)
C                           0   K=1
C
C THE IMAGE POLYGON MAY BE UNBOUNDED; PERMITTED ANGLES LIE IN THE
C RANGE -3.LE.BETAM(K).LE.1.    W(N) AND W(1) MUST BE FINITE.
C WE NORMALIZE BY THE CONDITIONS:
C
C    Z(N) = 1                                                   (2.1)
C    W(0.0) = WC  (A POINT IN THE INTERIOR OF THE POLYGON)      (2.1)
C
C CALLING SEQUENCE:
C
C   IPRINT  -2,-1,0, OR 1 FOR INCREASING AMOUNTS OF OUTPUT (INPUT)
C
C   IGUESS  1 IF AN INITIAL GUESS FOR Z IS SUPPLIED, OTHERWISE 0
C           (INPUT)
C
C   TOL     DESIRED ACCURACY IN SOLUTION OF NONLINEAR SYSTEM
C           (INPUT).  RECOMMENDED VALUE: 10.**(-NPTSQ-1) * TYPICAL
C           SIZE OF VERTICES W(K)
C
C   ERREST  ESTIMTATED ERROR IN SOLUTION (OUTPUT).  MORE
C           PRECISELY, ERREST IS AN APPROXIMATE BOUND FOR HOW FAR
C           THE TRUE VERTICES OF THE IMAGE POLYGON MAY BE FROM THOSE
C           COMPUTED BY NUMERICAL INTEGRATION USING THE
C           NUMERICALLY DETERMINED PREVERTICES Z(K).
C
C   N       NUMBER OF VERTICES OF THE IMAGE POLYGON (INPUT).
C           MUST BE .LE. 20
C
C   C       COMPLEX SCALE FACTOR IN FORMULA ABOVE (OUTPUT)
C
C   Z       COMPLEX ARRAY OF PREVERTICES ON THE UNIT CIRCLE.
C           DIMENSION AT LEAST N.  IF AN INITIAL GUESS IS
C           BEING SUPPLIED IT SHOULD BE IN Z ON INPUT, WITH Z(N)=1.
C           IN ANY CASE THE CORRECT PREVERTICES WILL BE IN Z ON OUTPUT.
C
C   WC      COMPLEX IMAGE OF 0 IN THE POLYGON, AS IN ABOVE FORMULA
C           (INPUT).  IT IS SAFEST TO PICK WC TO BE AS CENTRAL AS
C           POSSIBLE IN THE POLYGON IN THE SENSE THAT AS FEW PARTS
C           OF THE POLYGON AS POSSIBLE ARE SHIELDED FROM WC BY
C           REENTRANT EDGES.
C
C   W       COMPLEX ARRAY OF VERTICES OF THE IMAGE POLYGON
C           (INPUT).  DIMENSION AT LEAST N.  IT IS A GOOD IDEA
C           TO KEEP THE W(K) ROUGHLY ON THE SCALE OF UNITY.
C           W(K) WILL BE IGNORED WHEN THE VERTEX LIES AT INFINITY;
C           SEE BETAM, BELOW.  EACH CONNECTED BOUNDARY COMPONENT
C           MUST INCLUDE AT LEAST ONE VERTEX W(K), EVEN IF IT
C           HAS TO BE A DEGENERATE VERTEX WITH BETAM(K) = 0.
C           W(N) AND W(1) MUST BE FINITE.
C
C  BETAM    REAL ARRAY WITH BETAM(K) THE EXTERNAL ANGLE IN THE
C           POLYGON AT VERTEX K DIVIDED BY MINUS PI (INPUT).
```

```
C           DIMENSION AT LEAST N.  PERMITTED VALUES LIE IN
C           THE RANGE -3.LE.BETAM(K).LE.1.    (EXAMPLES: EACH
C           BETAM(K) IS -1/2 FOR A RECTANGLE, -2/3 FOR AN EQUI-
C           LATERAL TRIANGLE, +1 AT THE END OF A SLIT.)  THE
C           SUM OF THE BETAM(K) WILL BE -2 IF THEY HAVE BEEN
C           SET CORRECTLY.  BETAM(N-1) SHOULD NOT BE 0 OR 1.
C           W(K) LIES AT INFINITY IF AND ONLY IF BETAM(K).LE.-1.
C
C   NPTSQ   THE NUMBER OF POINTS TO BE USED PER SUBINTERVAL
C           IN GAUSS-JACOBI QUADRATURE (INPUT).  RECOMMENDED
C           VALUE: EQUAL TO ONE MORE THAN THE NUMBER OF DIGITS
C           OF ACCURACY DESIRED IN THE ANSWER.  MUST BE THE SAME
C           AS IN THE CALL TO QINIT WHICH FILLED THE VECTOR QWORK.
C
C   QWORK   REAL QUADRATURE WORK ARRAY (INPUT).  DIMENSION
C           AT LEAST NPTSQ * (2N+3) BUT NO GREATER THAN 460.
C           BEFORE CALLING SCSOLV QWORK MUST HAVE BEEN FILLED
C           BY SUBROUTINE QINIT.
C
C THE PROBLEM IS SOLVED BY FINDING THE
C SOLUTION TO A SYSTEM OF N-1 NONLINEAR EQUATIONS IN THE N-1
C UNKNOWNS Y(1),...,Y(N-1), WHICH ARE RELATED TO THE POINTS
C Z(K) BY THE FORMULA:
C
C      Y(K) = LOG ((TH(K)-TH(K-1))/(TH(K+1)-TH(K)))            (2.7)
C
C WHERE TH(K) DENOTES THE ARGUMENT OF Z(K).
C SUBROUTINE SCFUN DEFINES THIS SYSTEM OF EQUATIONS.
C THE ORIGINAL PROBLEM IS SUBJECT TO THE CONTRAINTS TH(K) < TH(K+1),
C BUT THESE VANISH IN THE TRANSFORMATION FROM Z TO Y.
C
C REFERENCE:  L. N. TREFETHEN, "NUMERICAL COMPUTATION OF THE
C   SCHWARZ-CHRISTOFFEL TRANSFORMATION," SIAM J. SCI. STAT. COMP. 1
C   (1980), 82-102.    EQUATION NOS. ABOVE ARE TAKEN FROM THIS PAPER.
C
C LLOYD N. TREFETHEN
C DEPARTMENT OF MATHEMATICS
C MASSACHUSETTS INSTITUTE OF TECHNOLOGY
C CAMBRIDGE, MA 02139
C (617) 253-4986 / LNT@MATH.MIT.EDU
C OCTOBER 1979 (VERSION 1); JULY 1983 (VERSION 2)
C
      IMPLICIT DOUBLE PRECISION (A-B,D-H,O-V,X-Y)
      IMPLICIT COMPLEX*16(C,W,Z)
      COMMON /PARAM1/ KFIX(20),KRAT(20),NCOMP,NPTSQ2,C2,
     &  QWORK2(460),BETAM2(20),Z2(20),WC2,W2(20)
      DIMENSION Z(N),W(N),BETAM(N),QWORK(1)
      DIMENSION AJINV(20,20),SCR(900),FVAL(19),Y(19)
      EXTERNAL SCFUN
      NM = N-1
C
C CHECK INPUT DATA:
      CALL CHECK(TOL,N,W,BETAM)
C
C DETERMINE NUMBER OF BOUNDARY COMPONENTS, ETC.:
C   PASS 1:  ONE FIXED POINT FOR EACH INFINITE VERTEX:
      NCOMP = 0
      DO 1 K = 2,NM
        IF (BETAM(K).GT.-1.D0) GOTO 1
        NCOMP = NCOMP + 1
        KFIX(NCOMP) = K - 1
        IF (NCOMP.EQ.1) KFIX(NCOMP) = 1
    1 CONTINUE
      IF (NCOMP.GT.0) GOTO 2
      NCOMP = 1
      KFIX(NCOMP) = 1
C   PASS 2:  ONE RATIO FOR EACH LINE SEGMENT:
    2 CONTINUE
      NEQ = 2*NCOMP
      DO 3 K = 1,NM
        IF (NEQ.EQ.N-1) GOTO 4
        IF (BETAM(K).LE.-1.D0.OR.BETAM(K+1).LE.-1.D0) GOTO 3
        NEQ = NEQ + 1
        KRAT(NEQ) = K
    3 CONTINUE
    4 Z(N) = (1.D0,0.D0)
C
C INITIAL GUESS, CASE IGUESS.EQ.0:
C (VERTICES EQUALLY SPACED AROUND CIRCLE):
      IF (IGUESS.NE.0) GOTO 11
      DO 5 K = 1,NM
    5   Y(K) = 0.D0
      GOTO 12
C
C INITIAL GUESS, CASE IGUESS.NE.0:
C (VERTICES SUPPLIED ON INPUT):
   11 CONTINUE
      DO 9 K = 1,NM
        KM = K-1
        IF (KM.EQ.0) KM = N
        TMP1 = DIMAG(LOG(Z(K+1)/Z(K)))
        IF (TMP1.LT.0.D0) TMP1 = TMP1 + 2.D0 * ACOS(-1.D0)
        TMP2 = DIMAG(LOG(Z(K)/Z(KM)))
        IF (TMP2.LT.0.D0) TMP2 = TMP2 + 2.D0 * ACOS(-1.D0)
    9   Y(K) = LOG(TMP2) - LOG(TMP1)
   12 CONTINUE
C
C NS01A CONTROL PARAMETERS:
      DSTEP = 1.D-6
      DMAX = 20.D0
      MAXFUN = (N-1) * 15
C
C COPY INPUT DATA TO /PARAM1/ FOR SCFUN:
C (THIS IS NECESSARY BECAUSE NS01A REQUIRES A FIXED CALLING
C SEQUENCE IN SUBROUTINE SCFUN.)
      NPTSQ2 = NPTSQ
      WC2 = WC
      DO 6 K = 1,N
        Z2(K) = Z(K)
        BETAM2(K) = BETAM(K)
    6   W2(K) = W(K)
      NWDIM = NPTSQ * (2*N+3)
      DO 7 I = 1,NWDIM
    7   QWORK2(I) = QWORK(I)
C
C SOLVE NONLINEAR SYSTEM WITH NS01A:
      CALL NS01A(NM,Y,FVAL,AJINV,DSTEP,DMAX,TOL,MAXFUN,
     &  IPRINT,SCR,SCFUN)
C
C COPY OUTPUT DATA FROM /PARAM1/:
      C = C2
      DO 8 K = 1,NM
    8   Z(K) = Z2(K)
C
C PRINT RESULTS AND TEST ACCURACY:
      IF (IPRINT.GE.0) CALL SCOUTP(N,C,Z,WC,W,BETAM,NPTSQ)
      CALL SCTEST(ERREST,N,C,Z,WC,W,BETAM,NPTSQ,QWORK)
      IF (IPRINT.GE.-1) WRITE (6,201) ERREST
  201 FORMAT (' ERREST:',E12.4/)
      RETURN
C
      END
```

```
C*********************************************************************
C* WSC                                           PRIMARY SUBROUTINE  **
C*********************************************************************
C
      FUNCTION WSC(ZZ,KZZ,Z0,W0,K0,N,C,Z,BETAM,NPTSQ,QWORK)
C
C COMPUTES FORWARD MAP W(ZZ)
C
C CALLING SEQUENCE:
C
C   ZZ      POINT IN THE DISK AT WHICH W(ZZ) IS DESIRED (INPUT)
C
C   KZZ     K IF ZZ = Z(K) FOR SOME K, OTHERWISE 0 (INPUT)
C
C   Z0      NEARBY POINT IN THE DISK AT WHICH W(Z0) IS KNOWN AND
C           FINITE (INPUT)
C
C   W0      W(Z0)  (INPUT)
C
C   K0      K IF Z0 = Z(K) FOR SOME K, OTHERWISE 0 (INPUT)
C
C   N,C,Z,BETAM,NPTSQ,QWORK     AS IN SCSOLV (INPUT)
C
C CONVENIENT VALUES OF Z0, W0, AND K0 FOR MOST APPLICATIONS CAN BE
C SUPPLIED BY SUBROUTINE NEARZ.
C
      IMPLICIT DOUBLE PRECISION (A-B,D-H,O-V,X-Y)
      IMPLICIT COMPLEX*16(C,W,Z)
      DIMENSION Z(N),BETAM(N),QWORK(1)
C
      WSC = W0 + C * ZQUAD(Z0,K0,ZZ,KZZ,N,Z,BETAM,NPTSQ,QWORK)
C
      RETURN
      END


C*********************************************************************
C* ZSC                                           PRIMARY SUBROUTINE  **
C*********************************************************************
C
      FUNCTION ZSC(WW,IGUESS,ZINIT,Z0,W0,K0,EPS,IER,N,C,
     &  Z,WC,W,BETAM,NPTSQ,QWORK)
C
C COMPUTES INVERSE MAP Z(WW) BY NEWTON ITERATION
C
C CALLING SEQUENCE:
C
C   WW      POINT IN THE POLYGON AT WHICH Z(WW) IS DESIRED (INPUT)
C
C   IGUESS (INPUT)
C           .EQ.1 - INITIAL GUESS IS SUPPLIED AS PARAMETER ZINIT
C           .NE.1 - GET INITIAL GUESS FROM PROGRAM ODE (SLOWER).
C                   FOR THIS THE SEGMENT WC-WW MUST LIE WITHIN
C                   THE POLYGON.
C
C   ZINIT   INITIAL GUESS IF IGUESS.EQ.1, OTHERWISE IGNORED (INPUT).
C           MAY NOT BE A PREVERTEX Z(K)
C
C   Z0      POINT IN THE DISK NEAR Z(WW) AT WHICH W(Z0) IS KNOWN AND
C           FINITE (INPUT).
C
C   W0      W(Z0)  (INPUT).  THE LINE SEGMENT FROM W0 TO WW MUST
C           LIE ENTIRELY WITHIN THE CLOSED POLYGON.
C
C   K0      K IF Z0 = Z(K) FOR SOME K, OTHERWISE 0 (INPUT)
C
C   EPS     DESIRED ACCURACY IN ANSWER Z(WW)  (INPUT)
```

```
C
C   IER     ERROR FLAG (INPUT AND OUTPUT).
C           ON INPUT, GIVE IER.NE.0 TO SUPPRESS ERROR MESSAGES.
C           ON OUTPUT, IER.NE.0 INDICATES UNSUCCESSFUL COMPUTATION --
C           TRY AGAIN WITH A BETTER INITIAL GUESS.
C
C   N,C,Z,WC,W,BETAM,NPTSQ,QWORK      AS IN SCSOLV (INPUT)
C
C CONVENIENT VALUES OF Z0, W0, AND K0 FOR SOME APPLICATIONS CAN BE
C SUPPLIED BY SUBROUTINE NEARW.
C
      IMPLICIT DOUBLE PRECISION (A-B,D-H,O-V,X-Y)
      IMPLICIT COMPLEX*16(C,W,Z)
      DIMENSION SCR(142),ISCR(5)
      DIMENSION Z(N),W(N),BETAM(N),QWORK(1)
      EXTERNAL ZFODE
      LOGICAL ODECAL
      COMMON /PARAM2/ CDWDT,Z2(20),BETAM2(20),N2
C
      ODECAL = .FALSE.
      IF (IGUESS.NE.1) GOTO 1
      ZI = ZINIT
      GOTO 3
C
C GET INITIAL GUESS ZI FROM PROGRAM ODE:
    1 N2 = N
      DO 2 K = 1,N
        Z2(K) = Z(K)
    2   BETAM2(K) = BETAM(K)
      ZI = (0.D0,0.D0)
      T = 0.D0
      IFLAG = -1
      RELERR = 0.D0
      ABSERR = 1.D-4
      CDWDT = (WW-WC)/C
      CALL ODE(ZFODE,2,ZI,T,1.D0,RELERR,ABSERR,IFLAG,SCR,ISCR)
      IF (IFLAG.NE.2.AND.IER.EQ.0) WRITE (6,201) IFLAG
      ODECAL = .TRUE.
C
C REFINE ANSWER BY NEWTON ITERATION:
    3 CONTINUE
      DO 4 ITER = 1,10
        ZFNWT = WW - WSC(ZI,0,Z0,W0,K0,N,C,Z,BETAM,NPTSQ,QWORK)
        ZI = ZI + ZFNWT/(C*ZPROD(ZI,0,N,Z,BETAM))
        IF (ABS(ZI).GE.1.1D0) ZI = .5D0 * ZI/ABS(ZI)
        IF (ABS(ZFNWT).LT.EPS) GOTO 5
    4   CONTINUE
      IF (.NOT.ODECAL) GOTO 1
      IF (IER.EQ.0) WRITE (6,202)
      IER = 1
    5 ZSC = ZI
C
  201 FORMAT (/' *** NONSTANDARD RETURN FROM ODE IN ZSC: IFLAG =',I2/)
  202 FORMAT (/' *** POSSIBLE ERROR IN ZSC: NO CONVERGENCE IN 10'/
     &       '      ITERATIONS.  MAY NEED A BETTER INITIAL GUESS ZINIT')
      RETURN
      END


C*******************************************************************
C* ZFODE                                SUBORDINATE(ZSC) SUBROUTINE  **
C*******************************************************************
C
      SUBROUTINE ZFODE(T,ZZ,ZDZDT)
C
C COMPUTES THE FUNCTION ZDZDT NEEDED BY ODE IN ZSC.
C
```

```
      IMPLICIT DOUBLE PRECISION (A-B,D-H,O-V,X-Y)
      IMPLICIT COMPLEX*16(C,W,Z)
      COMMON /PARAM2/ CDWDT,Z(20),BETAM(20),N
C
      ZDZDT = CDWDT / ZPROD(ZZ,0,N,Z,BETAM)
C
      RETURN
      END

C*******************************************************************
C* CHECK                            SUBORDINATE(SCSOLV) SUBROUTINE **
C*******************************************************************
C
      SUBROUTINE CHECK(EPS,N,W,BETAM)
C
C CHECKS GEOMETRY OF THE PROBLEM TO MAKE SURE IT IS A FORM USABLE
C BY SCSOLV.
C
      IMPLICIT DOUBLE PRECISION (A-B,D-H,O-V,X-Y)
      IMPLICIT COMPLEX*16(C,W,Z)
      DIMENSION W(N),BETAM(N)
C
      SUM = 0.D0
      DO 1 K = 1,N
    1   SUM = SUM + BETAM(K)
      IF (ABS(SUM+2.D0).LT.EPS) GOTO 2
      WRITE (6,301)
    2 IF (BETAM(1).GT.-1.D0) GOTO 3
      WRITE (6,302)
      STOP
    3 IF (BETAM(N).GT.-1.D0) GOTO 4
      WRITE (6,303)
      STOP
    4 IF (ABS(BETAM(N-1)).GT.EPS) GOTO 5
      WRITE (6,304)
      WRITE (6,306)
    5 IF (ABS(BETAM(N-1)-1.D0).GT.EPS) GOTO 6
      WRITE (6,305)
      WRITE (6,306)
      STOP
    6 DO 7 K = 2,N
        IF (BETAM(K).LE.-1.D0.OR.BETAM(K-1).LE.-1.D0) GOTO 7
        IF (ABS(W(K)-W(K-1)).LE.EPS) GOTO 8
    7   CONTINUE
      IF (ABS(W(1)-W(N)).GT.EPS) GOTO 9
    8 WRITE (6,307)
      STOP
    9 IF (N.GE.3) GOTO 10
      WRITE (6,309)
      STOP
   10 IF (N.LE.20) GOTO 11
      WRITE (6,310)
      STOP
   11 CONTINUE
      RETURN
C
  301 FORMAT (/' *** ERROR IN CHECK: ANGLES DO NOT ADD UP TO 2'/)
  302 FORMAT (/' *** ERROR IN CHECK: W(1) MUST BE FINITE'/)
  303 FORMAT (/' *** ERROR IN CHECK: W(N) MUST BE FINITE'/)
  304 FORMAT (/' *** WARNING IN CHECK: W(N-1) NOT DETERMINED'/)
  305 FORMAT (/' *** ERROR IN CHECK: W(N-1) NOT DETERMINED')
  306 FORMAT (/'   RENUMBER VERTICES SO THAT BETAM(N-1) IS NOT 0 OR 1')
  307 FORMAT (/' *** ERROR IN CHECK: TWO ADJACENT VERTICES ARE EQUAL'/)
  309 FORMAT (/' *** ERROR IN CHECK: N MUST BE NO LESS THAN 3'/)
  310 FORMAT (/' *** ERROR IN CHECK: N MUST BE NO MORE THAN 20'/)
      END
```

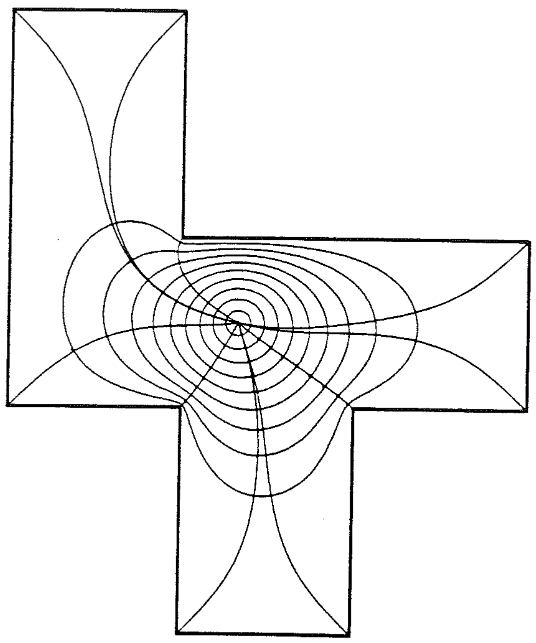
```
C************************************************************************
C* YZTRAN                             SUBORDINATE(SCSOLV) SUBROUTINE   **
C************************************************************************
C
      SUBROUTINE YZTRAN(N,Y,Z)
C
C TRANSFORMS Y(K) TO Z(K).  SEE COMMENTS IN SUBROUTINE SCSOLV.
C
      IMPLICIT DOUBLE PRECISION (A-B,D-H,O-V,X-Y)
      IMPLICIT COMPLEX*16(C,W,Z)
      DIMENSION Y(N),Z(N)
      NM = N - 1
      PI = ACOS(-1.D0)
C
      DTH = 1.D0
      THSUM = DTH
      DO 1 K = 1,NM
        DTH = DTH / EXP(Y(K))
    1   THSUM = THSUM + DTH
C
      DTH = 2.D0 * PI / THSUM
      THSUM = DTH
      Z(1) = DCMPLX(COS(DTH),SIN(DTH))
      DO 2 K = 2,NM
        DTH = DTH / EXP(Y(K-1))
        THSUM = THSUM + DTH
    2   Z(K) = DCMPLX(COS(THSUM),SIN(THSUM))
C
      RETURN
      END


C************************************************************************
C* SCFUN                              SUBORDINATE(SCSOLV) SUBROUTINE   **
C************************************************************************
      SUBROUTINE SCFUN(NDIM,Y,FVAL)
C
C THIS IS THE FUNCTION WHOSE ZERO MUST BE FOUND IN SCSOLV.
C
      IMPLICIT DOUBLE PRECISION (A-B,D-H,O-V,X-Y)
      IMPLICIT COMPLEX*16(C,W,Z)
      DIMENSION FVAL(NDIM),Y(NDIM)
      COMMON /PARAM1/ KFIX(20),KRAT(20),NCOMP,NPTSQ,C,
     &  QWORK(460),BETAM(20),Z(20),WC,W(20)
      N = NDIM+1
C
C TRANSFORM Y(K) TO Z(K):
      CALL YZTRAN(N,Y,Z)
C
C SET UP: COMPUTE INTEGRAL FROM 0 TO Z(N):
      WDENOM = ZQUAD((0.D0,0.D0),0,Z(N),N,N,Z,BETAM,NPTSQ,QWORK)
      C = (W(N)-WC) / WDENOM
C
C CASE 1:  W(K) AND W(K+1) FINITE:
C (COMPUTE INTEGRAL ALONG CHORD Z(K)-Z(K+1)):
      NFIRST = 2*NCOMP + 1
      IF (NFIRST.GT.N-1) GOTO 11
      DO 10 NEQ = NFIRST,NDIM
        KL = KRAT(NEQ)
        KR = KL+1
        ZINT = ZQUAD(Z(KL),KL,Z(KR),KR,N,Z,BETAM,NPTSQ,QWORK)
        FVAL(NEQ) = ABS(W(KR)-W(KL)) - ABS(C*ZINT)
   10 CONTINUE
C
C CASE 2:  W(K+1) INFINITE:
C (COMPUTE CONTOUR INTEGRAL ALONG RADIUS 0-Z(K)):
   11 DO 20 NVERT = 1,NCOMP
```

```
         KR = KFIX(NVERT)
         ZINT = ZQUAD((0.D0,0.D0),0,Z(KR),KR,N,Z,BETAM,NPTSQ,QWORK)
         ZFVAL = W(KR) - WC - C*ZINT
         FVAL(2*NVERT-1) = DREAL(ZFVAL)
         FVAL(2*NVERT) = DIMAG(ZFVAL)
   20 CONTINUE
      RETURN
C
      END

C*********************************************************************
C* SCOUTP                              SUBORDINATE(SCSOLV) SUBROUTINE  **
C*********************************************************************
C
      SUBROUTINE SCOUTP(N,C,Z,WC,W,BETAM,NPTSQ)
C
C PRINTS A TABLE OF K, W(K), TH(K), BETAM(K), AND Z(K).
C ALSO PRINTS THE CONSTANTS N, NPTSQ, WC, C.
C
      IMPLICIT DOUBLE PRECISION (A-B,D-H,O-V,X-Y)
      IMPLICIT COMPLEX*16(C,W,Z)
      DIMENSION Z(N),W(N),BETAM(N)
C
      WRITE (6,102) N, NPTSQ
      PI = ACOS(-1.D0)
      DO 1 K = 1,N
        THDPI = DIMAG(LOG(Z(K))) / PI
        IF (THDPI.LE.0.D0) THDPI = THDPI + 2.D0
        IF (BETAM(K).GT.-1.D0) WRITE (6,103) K,W(K),THDPI,BETAM(K),Z(K)
    1   IF (BETAM(K).LE.-1.D0) WRITE (6,104) K,THDPI,BETAM(K),Z(K)
      WRITE (6,105) WC,C
      RETURN
C
  102 FORMAT (/' PARAMETERS DEFINING MAP:',15X,'(N =',
     &  I3,')',6X,'(NPTSQ =',I3,')'//
     &  '  K',10X,'W(K)',10X,'TH(K)/PI',5X,'BETAM(K)',
     &  13X,'Z(K)'/
     &  ' ---',9X,'----',10X,'--------',5X,'--------',
     &  13X,'----'/)
  103 FORMAT (I3,'    (',F6.3,',',F6.3,')',F14.8,F12.5,
     &  3X,'(',F11.8,',',F11.8,')')
  104 FORMAT (I3,'         INFINITY    ',F14.8,F12.5,
     &  3X,'(',F11.8,',',F11.8,')')
  105 FORMAT (/' WC = (',E15.8,',',E15.8,')'/
     &         '  C = (',E15.8,',',E15.8,')'/)
      END

C*********************************************************************
C* SCTEST                              SUBORDINATE(SCSOLV) SUBROUTINE  **
C*********************************************************************
C
      SUBROUTINE SCTEST(ERREST,N,C,Z,WC,W,BETAM,NPTSQ,QWORK)
C
C TESTS THE COMPUTED MAP FOR ACCURACY.
C
      IMPLICIT DOUBLE PRECISION (A-B,D-H,O-V,X-Y)
      IMPLICIT COMPLEX*16(C,W,Z)
      DIMENSION Z(N),W(N),BETAM(N),QWORK(1)
C
C TEST LENGTH OF RADII:
      ERREST = 0.D0
      DO 10 K = 2,N
        IF (BETAM(K).GT.-1.D0) RADE = ABS(WC -
     &    WSC((0.D0,0.D0),0,Z(K),W(K),K,N,C,Z,BETAM,NPTSQ,QWORK))
        IF (BETAM(K).LE.-1.D0) RADE = ABS(WSC((.1D0,.1D0),0,
     &    Z(K-1),W(K-1),K-1,N,C,Z,BETAM,NPTSQ,QWORK)
     &     - WSC((.1D0,.1D0),0,Z(K+1),W(K+1),K+1,
     &     N,C,Z,BETAM,NPTSQ,QWORK))
          ERREST = MAX(ERREST,RADE)
   10   CONTINUE
        RETURN
        END


C*******************************************************************
C* ZQUAD                                         PRIMARY SUBROUTINE  **
C*******************************************************************
C
      FUNCTION ZQUAD(ZA,KA,ZB,KB,N,Z,BETAM,NPTSQ,QWORK)
C
C COMPUTES THE COMPLEX LINE INTEGRAL OF ZPROD FROM ZA TO ZB ALONG A
C STRAIGHT LINE SEGMENT WITHIN THE UNIT DISK.  FUNCTION ZQUAD1 IS
C CALLED TWICE, ONCE FOR EACH HALF OF THIS INTEGRAL.
C
      IMPLICIT DOUBLE PRECISION (A-B,D-H,O-V,X-Y)
      IMPLICIT COMPLEX*16(C,W,Z)
      DIMENSION Z(N),BETAM(N),QWORK(1)
C
      IF (ABS(ZA).GT.1.1D0.OR.ABS(ZB).GT.1.1D0) WRITE (6,301)
  301 FORMAT (/' *** WARNING IN ZQUAD: Z OUTSIDE THE DISK')
C
      ZMID = (ZA + ZB) / 2.D0
      ZQUAD = ZQUAD1(ZA,ZMID,KA,N,Z,BETAM,NPTSQ,QWORK)
     &  - ZQUAD1(ZB,ZMID,KB,N,Z,BETAM,NPTSQ,QWORK)
      RETURN
      END


C*******************************************************************
C* ZQUAD1                           SUBORDINATE(ZQUAD) SUBROUTINE  **
C*******************************************************************
C
      FUNCTION ZQUAD1(ZA,ZB,KA,N,Z,BETAM,NPTSQ,QWORK)
C
C COMPUTES THE COMPLEX LINE INTEGRAL OF ZPROD FROM ZA TO ZB ALONG A
C STRAIGHT LINE SEGMENT WITHIN THE UNIT DISK.  COMPOUND ONE-SIDED
C GAUSS-JACOBI QUADRATURE IS USED, USING FUNCTION DIST TO DETERMINE
C THE DISTANCE TO THE NEAREST SINGULARITY Z(K).
C
      IMPLICIT DOUBLE PRECISION (A-B,D-H,O-V,X-Y)
      IMPLICIT COMPLEX*16(C,W,Z)
      DIMENSION Z(N),BETAM(N),QWORK(1)
      DATA RESPRM /1.D0/
C
C CHECK FOR ZERO-LENGTH INTEGRAND:
      IF (ABS(ZA-ZB).GT.0.D0) GOTO 1
      ZQUAD1 = (0.D0,0.D0)
      RETURN
C
C STEP 1:  ONE-SIDED GAUSS-JACOBI QUADRATURE FOR LEFT ENDPOINT:
    1 R = MIN(1.D0,DIST(ZA,KA,N,Z)*RESPRM/ABS(ZA-ZB))
      ZAA = ZA + R*(ZB-ZA)
      ZQUAD1 = ZQSUM(ZA,ZAA,KA,N,Z,BETAM,NPTSQ,QWORK)
C
C STEP 2:  ADJOIN INTERVALS OF PURE GAUSSIAN QUADRATURE IF NECESSARY:
   10 IF (R.EQ.1.D0) RETURN
      R = MIN(1.D0,DIST(ZAA,0,N,Z)*RESPRM/ABS(ZAA-ZB))
      ZBB = ZAA + R*(ZB-ZAA)
      ZQUAD1 = ZQUAD1 + ZQSUM(ZAA,ZBB,0,N,Z,BETAM,NPTSQ,QWORK)
      ZAA = ZBB
      GOTO 10
      END
```

```
C*********************************************************************
C* DIST                              SUBORDINATE(ZQUAD) SUBROUTINE  **
C*********************************************************************
C
      FUNCTION DIST(ZZ,KS,N,Z)
C
C DETERMINES THE DISTANCE FROM ZZ TO THE NEAREST SINGULARITY Z(K)
C OTHER THAN Z(KS).
C
      IMPLICIT DOUBLE PRECISION (A-B,D-H,O-V,X-Y)
      IMPLICIT COMPLEX*16(C,W,Z)
      DIMENSION Z(N)
C
      DIST = 99.D0
      DO 1 K = 1,N
        IF (K.EQ.KS) GOTO 1
        DIST = MIN(DIST,ABS(ZZ-Z(K)))
    1   CONTINUE
      RETURN
      END

C*********************************************************************
C* ZQSUM                             SUBORDINATE(ZQUAD) SUBROUTINE  **
C*********************************************************************
C
      FUNCTION ZQSUM(ZA,ZB,KA,N,Z,BETAM,NPTSQ,QWORK)
C
C COMPUTES THE INTEGRAL OF ZPROD FROM ZA TO ZB BY APPLYING A
C ONE-SIDED GAUSS-JACOBI FORMULA WITH POSSIBLE SINGULARITY AT ZA.
C
      IMPLICIT DOUBLE PRECISION (A-B,D-H,O-V,X-Y)
      IMPLICIT COMPLEX*16(C,W,Z)
      DIMENSION Z(N),BETAM(N),QWORK(1)
C
      ZS = (0.D0,0.D0)
      ZH = (ZB-ZA) / 2.D0
      ZC = (ZA+ZB) / 2.D0
      K = KA
      IF (K.EQ.0) K = N+1
      IWT1 = NPTSQ*(K-1) + 1
      IWT2 = IWT1 + NPTSQ - 1
      IOFFST = NPTSQ*(N+1)
      DO 1 I = IWT1,IWT2
    1   ZS = ZS + QWORK(IOFFST+I)*ZPROD(ZC+ZH*QWORK(I),KA,N,Z,BETAM)
      ZQSUM = ZS*ZH
      IF (ABS(ZH).NE.0.D0.AND.K.NE.N+1)
     &  ZQSUM = ZQSUM*ABS(ZH)**BETAM(K)
      RETURN
      END

C*********************************************************************
C* ZPROD                             SUBORDINATE(ZQUAD) SUBROUTINE  **
C*********************************************************************
C
      FUNCTION ZPROD(ZZ,KS,N,Z,BETAM)
C
C COMPUTES THE INTEGRAND
C
C            N
C          PROD  (1-ZZ/Z(K))**BETAM(K)   ,
C           K=1
C
C TAKING ARGUMENT ONLY (NOT MODULUS) FOR TERM K = KS.
C
C *** NOTE -- IN PRACTICE THIS IS THE INNERMOST SUBROUTINE
C *** IN SCPACK CALCULATIONS.  THE COMPLEX LOG CALCULATION BELOW
```

```
C *** MAY ACCOUNT FOR AS MUCH AS HALF OF THE TOTAL EXECUTION TIME.
C
      IMPLICIT DOUBLE PRECISION (A-B,D-H,O-V,X-Y)
      IMPLICIT COMPLEX*16(C,W,Z)
      DIMENSION Z(N),BETAM(N)
      COMMON /LOGCNT/ NCOUNT,NCNT1
C
      ZSUM = (0.D0,0.D0)
      DO 1 K = 1,N
        ZTMP = (1.D0,0.D0) - ZZ/Z(K)
        IF (K.EQ.KS) ZTMP = ZTMP / ABS(ZTMP)
    1   ZSUM = ZSUM + BETAM(K)*LOG(ZTMP)
      ZPROD = EXP(ZSUM)
      NCOUNT = NCOUNT + N
      RETURN
      END

C**********************************************************************
C* RPROD                                         PRIMARY SUBROUTINE  **
C**********************************************************************
C
      FUNCTION RPROD(ZZ,N,Z,BETAM)
C
C COMPUTES THE ABSOLUTE VALUE OF THE INTEGRAND
C
C           N
C         PROD  (1-ZZ/Z(K))**BETAM(K)
C          K=1
C
      IMPLICIT DOUBLE PRECISION (A-B,D-H,O-V,X-Y)
      IMPLICIT COMPLEX*16(C,W,Z)
      DIMENSION Z(N),BETAM(N)
C
      SUM = 0.D0
      DO 1 K = 1,N
        ZTMP = (1.D0,0.D0) - ZZ/Z(K)
    1   SUM = SUM + BETAM(K)*LOG(ABS(ZTMP))
      RPROD = EXP(SUM)
      RETURN
      END

C**********************************************************************
C* NEARZ                                         PRIMARY SUBROUTINE  **
C**********************************************************************
C
      SUBROUTINE NEARZ(ZZ,ZN,WN,KN,N,Z,WC,W,BETAM)
C
C RETURNS INFORMATION ASSOCIATED WITH THE NEAREST PREVERTEX Z(K)
C TO THE POINT ZZ, OR WITH 0 IF 0 IS CLOSER THAN ANY Z(K).
C ZN = PREVERTEX POSITION, WN = W(ZN), KN = PREVERTEX NO. (0 TO N)
C
      IMPLICIT DOUBLE PRECISION (A-B,D-H,O-V,X-Y)
      IMPLICIT COMPLEX*16(C,W,Z)
      DIMENSION Z(N),W(N),BETAM(N)
C
      DIST = ABS(ZZ)
      KN = 0
      ZN = (0.D0,0.D0)
      WN = WC
      IF (DIST.LE..5D0) RETURN
      DO 1 K = 1,N
        IF (BETAM(K).LE.-1.D0) GOTO 1
        DISTZK = ABS(ZZ-Z(K))
        IF (DISTZK.GE.DIST) GOTO 1
        DIST = DISTZK
        KN = K
    1   CONTINUE
        IF (KN.EQ.0) RETURN
        ZN = Z(KN)
        WN = W(KN)
        RETURN
        END

C***********************************************************************
C* NEARW                                          PRIMARY SUBROUTINE  **
C***********************************************************************
C
      SUBROUTINE NEARW(WW,ZN,WN,KN,N,Z,WC,W,BETAM)
C
C RETURNS INFORMATION ASSOCIATED WITH THE NEAREST VERTEX W(K)
C TO THE POINT WW, OR WITH WC IF WC IS CLOSER THAN ANY W(K).
C ZN = PREVERTEX POSITION, WN = W(ZN), KN = VERTEX NO. (0 TO N)
C
      IMPLICIT DOUBLE PRECISION (A-B,D-H,O-V,X-Y)
      IMPLICIT COMPLEX*16(C,W,Z)
      DIMENSION Z(N),W(N),BETAM(N)
C
      DIST = ABS(WW-WC)
      KN = 0
      ZN = (0.D0,0.D0)
      WN = WC
      DO 1 K = 1,N
        IF (BETAM(K).LE.-1.D0) GOTO 1
        DISTWK = ABS(WW-W(K))
        IF (DISTWK.GE.DIST) GOTO 1
        DIST = DISTWK
        KN = K
    1   CONTINUE
      IF (KN.EQ.0) RETURN
      ZN = Z(KN)
      WN = W(KN)
      RETURN
      END

C***********************************************************************
C* ANGLES                                         PRIMARY SUBROUTINE  **
C***********************************************************************
C
      SUBROUTINE ANGLES(N,W,BETAM)
C
C COMPUTES EXTERNAL ANGLES -PI*BETAM(K) FROM KNOWLEDGE OF
C THE VERTICES W(K).  AN ANGLE BETAM(K) IS COMPUTED FOR EACH
C K FOR WHICH W(K-1), W(K), AND W(K+1) ARE FINITE.
C TO GET THIS INFORMATION ACROSS ANY VERTICES AT INFINITY
C SHOULD BE SIGNALED BY THE VALUE W(K) = (99.,99.)  ON INPUT.
C
      IMPLICIT DOUBLE PRECISION (A-B,D-H,O-V,X-Y)
      IMPLICIT COMPLEX*16(C,W,Z)
      DIMENSION W(N),BETAM(N)
      C9 = (99.D0,99.D0)
C
      PI = ACOS(-1.D0)
      DO 1 K = 1,N
        KM = MOD(K+N-2,N)+1
        KP = MOD(K,N)+1
        IF (W(KM).EQ.C9.OR.W(K).EQ.C9.OR.W(KP).EQ.C9) GOTO 1
        BETAM(K) = DIMAG(LOG((W(KM)-W(K))/(W(KP)-W(K))))/PI - 1.D0
        IF (BETAM(K).LE.-1.D0) BETAM(K) = BETAM(K) + 2.D0
    1   CONTINUE
      RETURN
      END
```

```
C***********************************************************************
C* COUNT0                                        PRIMARY SUBROUTINE  **
C***********************************************************************
C
C INITIALIZES THE LOGARITHM COUNTER (SEE FUNCTION ZPROD)
C
      SUBROUTINE COUNT0
      COMMON /LOGCNT/ NCOUNT,NCNT1
      NCOUNT = 0
      NCNT1 = 0
      WRITE (6,1)
    1 FORMAT (' ------- LOG COUNTER SET TO ZERO')
      RETURN
      END


C***********************************************************************
C* COUNT                                         PRIMARY SUBROUTINE  **
C***********************************************************************
C
C PRINTS THE NUMBER OF LOGARITHMS SINCE THE LAST CALL
C
      SUBROUTINE COUNT
      COMMON /LOGCNT/ NCOUNT,NCNT1
      NCDIFF = NCOUNT - NCNT1
      WRITE (6,2) NCDIFF,NCOUNT
    2 FORMAT (' ------- NO. LOGS: SINCE LAST COUNT',I7,',   TOTAL',I8)
      NCNT1 = NCOUNT
      RETURN
      END
```



7.2. RESIST (conformal modulus of a quadrilateral)

```
      FUNCTION RESIST(N,W,WC,IBRK,NDIG,ERREST,QWORK)
C
C THIS FUNCTION RETURNS THE RESISTANCE (CONFORMAL MODULUS)
C OF A POLYGONALLY SHAPED RESISTOR ("QUADRILATERAL").
C COMPUTATIONS ARE BASED ON THE SCHWARZ-CHRISTOFFEL TRANSFORMATION.
C WE NORMALIZE BY ASSUMING THAT A SQUARE HAS RESISTANCE 1.
C
C INPUT PARAMETERS:
C
C   N       NUMBER OF VERTICES OF THE POLYGON.  N MUST SATISFY
C           4 .LE. N .LE. 20.
C
C   W       COMPLEX ARRAY OF DIMENSION AT LEAST N CONTAINING
C           THE POSITIONS OF THE VERTICES, VIEWED AS COMPLEX NUMBERS.
C           THE VERTICES MUST BE LISTED IN COUNTERCLOCKWISE ORDER
C           AROUND THE POLYGON.  IT IS A GOOD IDEA TO KEEP THE
C           W(K) ROUGHLY ON THE SCALE OF UNITY.
C
C   WC      A POINT IN THE INTERIOR OF THE POLYGON.  TRY TO
C           PICK WC AS CENTRAL AS POSSIBLE IN THE SENSE THAT
C           AS LITTLE OF THE POLYGON AS POSSIBLE IS SHIELDED
C           FROM IT BY REENTRANT EDGES.
C
C   IBRK    ARRAY OF DIMENSION AT LEAST 4 CONTAINING INDICES OF
C           THE VERTICES WHICH DEFINE THE BREAKS
C           BETWEEN CONSTANT-VOLTAGE AND INSULATED PORTIONS
C           OF THE BOUNDARY.  THE PROGRAM WILL ASSUME THAT
C           THE VOLTAGE IS APPLIED BETWEEN SIDE W(IBRK(1))-W(IBRK(2))
C           AND SIDE W(IBRK(3))-W(IBRK(4)), WITH THE OTHER TWO
C           SIDES INSULATED.  THE BREAK VERTICES MUST BE NUMBERED
C           IN COUNTERCLOCKWISE ORDER; THUS THE INTEGERS IBRK(I)
C           MUST INCREASE WITH I, EXCEPT THAT THEY MAY WRAP
C           ONCE AROUND N.
C
C   NDIG    INPUT INTEGER GIVING THE DESIRED NUMBER OF DIGITS
C           OF ACCURACY IN THE RESULT.  NDIG MUST BE AT LEAST 2.
C           IT SHOULD BE NO GREATER THAN A COUPLE OF DIGITS LESS
C           THAN FULL-WORD PRECISION.
C
C OUTPUT PARAMETER:
C
C  ERREST   ROUGH BUT CONSERVATIVE ESTIMATE OF THE SIZE OF
C           THE ERROR IN THE VALUE RETURNED.
C
C WORK SPACE PARAMETER:
C
C   QWORK   REAL WORK ARRAY.  DIMENSION AT LEAST (NDIG+1)*(2N+3),
C           BUT NO MORE THAN 460.
C
C
C SOME ADVICE:
C
C   THE PROGRAM IS NOT INFALLIBLE.  IF IT YIELDS STRANGE
C   MESSAGES ABOUT NOT CONVERGING, IT'S PROBABLY BEST TO
C   TRY A SIMPLER GEOMETRY.  BECAUSE OF THE PROBLEM OF "CROWDING",
C   THE CALCULATION WILL PROBABLY FAIL FOR REGIONS WITH RESISTANCE
C   MUCH GREATER THAN 10.
C
C   THE AMOUNT OF TIME REQUIRED IS ROUGHLY PROPORTIONAL TO NDIG
C   AND ALSO ROUGHLY PROPORTIONAL TO N**3.
C   THUS PROBLEMS WITH MANY CORNERS CAN
C   TAKE QUITE A WHILE -- SEVERAL MINUTES OF CPU TIME ON
C   THE IBM 370-168 FOR A PROBLEM WITH N = 20.
C
```

```
C LLOYD N. TREFETHEN
C DEPARTMENT OF MATHEMATICS
C MASSACHUSETTS INSTITUTE OF TECHNOLOGY
C NOVEMBER 1979, REVISED JULY 1983
C
      IMPLICIT COMPLEX*16(C,W,Z), REAL*8(A-B,D-H,O-V,X-Y)
C     MAP FROM DISK TO RESISTOR:
        DIMENSION Z(20),W(1),IBRK(1),BETAM(20),QWORK(1)
C     MAP FROM DISK TO RECTANGLE WITH EQUAL RESISTANCE:
        DIMENSION Z2(4),W2(4),BETAM2(4)
C
      ZERO = (0.D0,0.D0)
      PI = ACOS(-1.D0)
C
C COMPUTE ANGLES AND CHECK INPUT PARAMETERS:
      CALL ANGLES(N,W,BETAM)
      IF (NDIG.LT.2) WRITE (6,101)
      IF (NDIG.LT.2) STOP
      IF (N.LT.4) WRITE (6,102)
      IF (N.LT.4) STOP
      DO 1 K = 1,4
        IF (IBRK(K).GT.N .OR. IBRK(K).LT.1) GOTO 2
    1   CONTINUE
      GOTO 3
    2 WRITE (6,103)
      STOP
    3 CONTINUE
C
C SET UP FIRST MAP:
      NPTSQ = NDIG
      TOL = MAX(1.E-12, 10.D0**(-NPTSQ-1))
      CALL QINIT(N,BETAM,NPTSQ,QWORK)
      CALL SCSOLV(-2,0,TOL,EEST,N,C,Z,WC,W,BETAM,NPTSQ,QWORK)
C
C SET UP MAP TO RECTANGLE:
C (QWORK IS OVERWRITTEN TO SAVE SPACE)
      N2 = 4
      C2 = (1.D0,0.D0)
      DO 9 K = 1,4
        BETAM2(K) = -.5D0
    9   Z2(K) = Z(IBRK(K))
      NPTSQ2 = NDIG + 1
      CALL QINIT(N2,BETAM2,NPTSQ2,QWORK)
      DO 12 K = 1,4
   12   W2(K)=WSC(Z2(K),K,ZERO,ZERO,0,N2,C2,Z2,BETAM2,NPTSQ2,QWORK)
C
C COMPUTE LENGTH, WIDTH, RESISTANCE, AND ERROR ESTIMATE:
      XLEN1 = ABS(W2(2)-W2(3))
      XLEN2 = ABS(W2(4)-W2(1))
      ERRL = MAX(ABS(XLEN1-XLEN2),2.D0*EEST) / XLEN1
      XWID1 = ABS(W2(2)-W2(1))
      XWID2 = ABS(W2(4)-W2(3))
      ERRW = MAX(ABS(XWID1-XWID2),2.D0*EEST) / XWID1
      RESIST = (XLEN1+XLEN2) / (XWID1+XWID2)
      ERREST = RESIST * (ERRL + ERRW)
C
      RETURN
C
  101 FORMAT (' *** ERROR IN RESIST *** NDIG SHOULD BE AT',
     &  ' LEAST 2.'/)
  102 FORMAT (' *** ERROR IN RESIST *** N MUST BE NO',
     &  ' GREATER THAN 20'/' AND NO SMALLER THAN 4')
  103 FORMAT (' *** ERROR IN RESIST *** EACH IBRK(I) MUST',
     &  ' BE IN THE',/,' RANGE FROM 1 TO N')
      END
```