

Appendix A. Original FORTRAN 66 code

For reference, this appendix includes the original FORTRAN 66 code written by A.B.Schubert in 1972 [4].

This code cannot be compiled and run directly, one reason being that physical punch cards are required.

```
C   Appendix: FORTRAN Program for Secondary Flow, by A. B. Schubert (1972)
C
C   TUBEFLOW   A PROGRAM WHICH COMPUTES THE SECONDARY
C   (CROSS-SECTIONAL) FLOW OF A FLUID IN A CURVED TUBE AND THE
C   VELOCITY AND FLUX IN THE AXIAL DIRECTION OF THE TUBE BY
C   DISCRETIZATION OF THE CROSS-SECTION AND OF THE GOVERNING
C   DIFFERENTIAL EQUATIONS.

PARAMETER NR=10,NA=18,NRP1=NR+1,NAP1=NA+1,NRM1=NR-1,NAM1=NA-1,
* NAH=NA/2+1

C       NR = NUMBER OF GRID SPACES ON (0,1) IN RADIAL (R) DIRECTION.
C       NA = NUMBER OF GRID SPACES ON (0,PI) IN ANGULAR (ALPHA)
C           DIRECTION.

C   PRINCIPAL DISCRETE FUNCTIONS COMPUTED ARE DEFINED AS FOLLOWS.
C   PHI(I,J,K)= NON-DIMENSIONAL STREAM FUNCTION VALUE FOR SECONDARY
C               FLOW AT POLAR GRID POINT CORRESPONDING TO I-TH
C               RADIAL VALUE (WHERE I= 1 CORRESPONDS TO R = 0 AND
C               I = NR+1 CORRESPONDS TO R = 1) AND J-TH ANGULAR
C               VALUE (WHERE J = 1 CORRESPONDS TO ALPHA = 0 AND
C               J = NA+1 CORRESPONDS TO ALPHA = PI.
C               K = 1 IMPLIES A VALUE AT THE PREVIOUS OUTER
C                   ITERATION.
C               K = 2 IMPLIES A VALUE AT THE PREVIOUS SOR ITERATION
C               K = 3 IMPLIES A VALUE AT THE CURRENT SOR AND OUTER
C                   ITERATION.
C   W(I,J,K) = NON-DIMENSIONAL VELOCITY IN THE AXIAL DIRECTION,
C               WITH SUBSCRIPTS DEFINED AS FOR PHI.
C   OMEGA(I,J,K) = INTERMEDIATE VARIABLE INTRODUCED IN ANALYTICAL
C                 DESCRIPTION OF PROBLEM, WITH SUBSCRIPTS DEFINED
C                 AS FOR PHI.

DIMENSION PHI(NRP1,NAP1,4),W(NRP1,NAP1,4),OMEGA(NRP1,NAP1,4),
* XI(4),RHO(3),EPS(3),SA(NAP1),COSA(NAP1),RINV(NRP1),RINV2(NRP1),
* RHOC(3),B(NRP1,5),C(NRP1,NAP1),EPPS(3),XIC(4),E(NRP1,NAP1,6),
* EE(NRP1),EF(NRP1),CA(NRP1,NAP1),DELA(NR),CO(NR),DOR(5)

DATA PI/3.14159255/,MAXSOR,MAXOUT/250,60/,NOR/3/,(DOR(I),I=1,3)
*/.2,.2,.2/,ISTART/1/

C   MAXSOR = MAXIMUM NUMBER OF ITERATIONS TO BE ALLOWED FOR COMPUTING
C           SOLUTION TO ANY SYSTEM BY SUCCESSIVE OVER-RELAXATION
C   MAXOUT = MAXIMUM NUMBER OF OUTER ITERATIONS TO BE ALLOWED IN THE
C           COMPUTATION OF PHI, W, AND OMEGA.
C   NOR    = NUMBER OF DIFFERENT OVER-RELAXATION FACTORS TO BE TRIED
C           IN ANY DATA CASE.
C   DOR(I) = AMOUNT OF DECREASE IN OVER-RELAXATION FACTOR AT I-TH
C           CHANGE.
C   ISTART = INPUT CONTROL VARIABLE INDICATING WHETHER OR NOT THE
C           FIRST OUTER ITERATES ARE TO BE READ FROM CARDS.
C           ISTART.EQ.0 IMPLIES STARTING VALUES WILL BE SET = 0 BY
C           PROGRAM.
```

```
C          ISTART.NE.0 IMPLIES STARTING ITERATES WILL BE READ FROM
C          CARDS IN THE FORMAT (8E10.5).

C      TEST FOR NECESSITY OF READING STARTING OUTER ITERATES FOR PHI, W,
C      AND OMEGA.
C      DSTART = VALUE OF PARAMETER D FOR WHICH THE STARTING VALUES
C      ARE A SOLUTION.

      IF(ISTART.NE.0) READ(5,88) DSTART,((PHI(I,J,4),J=1,NAP1),I=1,NRP1)
      ,((W(I,J,4),J=1,NAP1),I=1,NRP1),((OMEGA(I,J,4),J=1,NAP1),I=1,NRP1)
      KRP1=NRP1

C      DR = GRID SPACING IN RADIAL DIRECTION.
C      DA = GRID SPACING IN ANGULAR DIRECTION.

      DR=1./NR
      DA=PI/NA
      DAH=.5*DA
      DRH=.5*DR
      DRDAM=DR*DA
      DRDA=DR/DA
      DADR=DA/DR

C      COMPUTE ELEMENTS OF AREA, DELA(I),I=1,...,NR, IN RADIAL DIRECTION
C      FOR USE IN FLUX CALCULATION.

      DO 1 I=1,NR
1  DELA(I)=(2*I-1)*DRH*DRDAM

C      ON THE GRID DEFINED BY DR AND DA, COMPUTE DISCRETE FUNCTIONS
C      DEPENDENT ONLY ON RADIUS AND ANGLE.

      SA(1)=0.
      DO 2 J=2,NA
      SA(J)=SIN((J-1)*DA)*DAH
2  COSA(J)=COS((J-1)*DA)
      SA(NAP1)=0.
      COSA(NAP1)=-1.
      RINV(NRP1)=1.
      RINV2(NRP1)=1.
      DO 3 I=2,NR
      RINV(I)=1./((I-1)*DR)
      DRRH=DRH*RINV(I)
      RINV2(I)=RINV(I)**2
      DO 3 J=2,NA
3  CA(I,J)=DRRH*COSA(J)

C      COMPUTE SOR COEFFICIENTS FOR PHI.

      DO 4 I=2,NR
      BO=2.*(DADR+DRDA*RINV2(I))+DA*RINV(I)
      B(I,1)=(DADR+DA*RINV(I))/BO
      B(I,2)=DRDA*RINV2(I)/BO
```

```
B(I,3)=DADR/BO
B(I,4)=B(I,2)
4 B(I,5)=DRDAM/BO

C READ OUTER ITERATION SMOOTHING PARAMETERS (XI), OVER-RELAXATION
C FACTORS (RHO), OUTER ITERATION CONVERGENCE TOLERANCES (EPS),
C PARAMETER RELATED TO REYNOLDS NO. (D), AND INPUT/OUTPUT CONTROL
C VARIABLES DEFINED AS FOLLOWS.
C ISAVE.NE.0 IMPLIES SAVE THE SOLUTION FOR THIS CASE (IF OBTAINED)
C IN MEMORY FOR USE AS STARTING VALUES FOR A SUCCEEDING
C CASE WITH A DIFFERENT VALUE OF D.
C ISAVE.EQ.0 IMPLIES DO NOT DO THE ABOVE.
C IUSE.NE.0 IMPLIES TAKE STARTING VALUES FOR OUTER ITERATION FROM
C MEMORY.
C IUSE.EQ.0 IMPLIES INITIALIZE OUTER ITERATES TO ZERO.
C IPCH.NE.0 IMPLIES PUNCH SOLUTION FOR THIS CASE (IF OBTAINED)
C OUT ON CARDS IN THE FORMAT (8E10.5).
C IPCH.EQ.0 IMPLIES DO NOT PUNCH SOLUTION OBTAINED FOR THIS CASE.

5 READ(5,99,END=70) XI,RHO,EPS,D,ISAVE,IUSE,IPCH

C PRINT OUT INPUT PARAMETERS.

WRITE(6,98) D,RHO,XI,EPS

C COMPUTE PARAMETERS DEPENDENT ON THE INPUT PARAMETER D, INCLUDING
C THE COEFFICIENTS, CO(I),I=1,...,NR, IN THE NUMERICAL INTEGRATION
C FOR THE FLUX.

DDRDM=D*DRDAM
DDR2=D*DR**2
CON=4./(PI*D)
CO(1)=16.*DELA(1)/(3.*PI*D)
DO 105 I=2,NR
105 CO(I)=CON*DELA(I)

C COMPUTE COMPLEMENTS (RELATIVE TO 1) OF SMOOTHING PARAMETERS AND
C OVER-RELAXATION FACTORS.
C ALSO COMPUTE SOR CONVERGENCE TOLERANCES, EPPS(I),I=1,2,3, AS
C FUNCTIONS OF OUTER ITERATION TOLERANCES, EPS(I),I=1,2,3.

DO 6 I=1,3
XIC(I)=1.-XI(I)
RHOC(I)=1.-RHO(I)
6 EPPS(I)=.05*EPS(I)
XIC(4)=1.-XI(4)

C TEST WHETHER INITIAL OUTER ITERATES ARE TO BE OBTAINED FROM
C MEMORY, HAVING BEEN PLACED THERE AS THE SOLUTION FOR A PREVIOUS
C VALUE OF D EITHER BY CARD INPUT OR BY A PREVIOUS COMPUTATION
C THIS RUN.

IF(IUSE.EQ.0) GO TO 7
```

```
      DO 106 I=1,NRP1
      DO 106 J=1,NAP1
      PHI(I,J,3)=PHI(I,J,4)
      W(I,J,3)=W(I,J,4)
106  OMEGA(I,J,3)=OMEGA(I,J,4)
      WRITE(6,87) DSTART
      GO TO 9

C      INITIALIZE OUTER ITERATES TO ZERO IF NEITHER INPUT NOR COMPUTED
C      PREVIOUSLY THIS RUN.

      7 DO 8 I=1,NRP1
      DO 8 J=1,NAP1
      PHI(I,J,3)=0.
      W(I,J,3)=0.
      8 OMEGA(I,J,3)=0.

C      INITIALIZE OUTER ITERATION COUNTER (IOUT) AND COUNTERS OF NUMBER
C      OF OVER-RELAXATION FACTORS USED FOR W AND OMEGA (IRW,IRO, RESP.)
C      FOR THIS DATA CASE.

      9 IOUT=0
      IRW=0
      IRO=0

C      TEST FOR MAXIMUM NUMBER OF OUTER ITERATIONS.

10  IF(IOUT.GE.MAXOUT) GO TO 58

C      UPDATE OUTER ITERATION COUNTER, WRITE MESSAGE INDICATING NEW
C      OUTER ITERATION NUMBER, AND SET OUTER ITERATION CONVERGENCE
C      INDICATOR (ICV) TO ZERO. AFTER COMPUTATION OF ALL CURRENT OUTER
C      ITERATES AND COMPARISON WITH PREVIOUS OUTER ITERATES AGAINST THE
C      SPECIFIED TOLERANCES, ICV WILL STILL BE ZERO IF CONVERGENCE HAS
C      BEEN OBTAINED, OTHERWISE ICV WILL BE NON-ZERO.

      IOUT=IOUT+1
      WRITE(6,92) IOUT
      ICV=0

C      UPDATE PREVIOUS OUTER ITERATES.

      DO 12 I=1,NRP1
      DO 12 J=1,NAP1
      PHI(I,J,1)=PHI(I,J,3)
      W(I,J,1)=W(I,J,3)
12  OMEGA(I,J,1)=OMEGA(I,J,3)

C      COMPUTE CONSTANT SOR COEFFICIENT FOR PHI.

      DO 13 I=2,NR
      DO 13 J=2,NA
13  C(I,J)=B(I,5)*OMEGA(I,J,1)
```

```
C    INITIALIZE SOR ITERATION COUNTER FOR PHI.

      ISOR=0

C    TEST FOR MAXIMUM NUMBER OF SOR ITERATIONS FOR PHI.

14  IF(ISOR.GE.MAXSOR) GO TO 55

C    UPDATE SOR ITERATION COUNTER IF MAXIMUM NUMBER HAS NOT BEEN
C    ACHIEVED.

      ISOR=ISOR+1

C    UPDATE PREVIOUS SOR ITERATES FOR PHI.

      DO 16 I=2,NR
      DO 16 J=2,NA
16  PHI(I,J,2)=PHI(I,J,3)

C    SET SOR ITERATION CONVERGENCE INDICATOR (ICONV) TO ZERO. AFTER
C    COMPUTATION OF CURRENT SOR ITERATE AND COMPARISON WITH PREVIOUS
C    ITERATE AGAINST THE TOLERANCE, ICONV WILL STILL BE ZERO IF
C    CONVERGENCE HAS BEEN OBTAINED. OTHERWISE IT WILL BE NON-ZERO.

      ICONV=0

C    COMPUTE CURRENT SOR ITERATE FOR PHI...
C    ...FIRST ON INTERIOR OF REGION, EXCLUDING 'INNER BOUNDARY'
C    R = 1.-DR.

      DO 18 I=2,NRM1
      DO 18 J=2,NA
      PHI(I,J,2)=RHOC(1)*PHI(I,J,2)+RHO(1)*(B(I,1)*PHI(I+1,J,2)
* +B(I,2)*PHI(I,J+1,2)+B(I,3)*PHI(I-1,J,3)+B(I,4)*PHI(I,J-1,3)
* +C(I,J))
      IF(ABS(PHI(I,J,2)-PHI(I,J,3)).GT.EPPS(1)) ICONV=1
18  CONTINUE

C    ...THEN ON 'INNER BOUNDARY' R = 1.-DR.

      DO 19 J=2,NA
      PHI(NR,J,3)=RHOC(1)*PHI(NR,J,2)+RHO(1)*.25*PHI(NRM1,J,3)
      IF(ABS(PHI(NR,J,2)-PHI(NR,J,3)).GT.EPPS(1)) ICONV=1
19  CONTINUE

C    TEST FOR CONVERGENCE OF SOR ITERATION FOR PHI.

      IF(ICONV.NE.0) GO TO 14

C    SOR CONVERGENCE ATTAINED FOR PHI. SMOOTH OUTER ITERATE.

      DO 20 I=2,NR
```

```

DO 20 J=2,NA
  PHI(I,J,3)=XI(1)*PHI(I,J,1)+XIC(1)*PHI(I,J,3)
  IF(ABS(PHI(I,J,1)-PHI(I,J,3)).GT.EPS(1)) ICV=1
20 CONTINUE

C      PRINT OUT SMOOTHED CURRENT OUTER ITERATE FOR PHI.

      CALL OUTPUT('PHI',ISOR,PHI(1,1,3))

C      COMPUTE COEFFICIENTS FOR SOR ITERATION FOR W...
C      ...FIRST COMPUTE COEFFICIENTS AT THE ORIGIN

E0=4.+ABS(PHI(2,NAH,3))
E1=(1.-AMIN1(PHI(2,NAH,3),0.))/E0
E2=2./E0
E3=(1.+AMAX1(PHI(2,NAH,3),0.))/E0
E4=DDR2/E0

C      ...NEXT COMPUTE COEFFICIENTS FOR REMAINDER OF REGION...
C      FIRST FOR RAYS ALONG ALPHA = 0 AND ALPHA = PI, EXCLUDING
C      R = 0 AND R = 1.

DO 23 I=2,NR
  DELTA1=DA-PHI(I,2,3)
  DELTA2=DA+PHI(I,NA,3)
  EE(I)=2.*(DADR+DRDA*RINV2(I))
  EE1=EE(I)+RINV(I)*ABS(DELTA1)
  EE2=EE(I)+RINV(I)*ABS(DELTA2)
  E(I,1,1)=(DADR+RINV(I)*AMAX1(DELTA1,0.))/EE1
  E(I,NAP1,1)=(DADR+RINV(I)*AMAX1(DELTA2,0.))/EE2
  EF(I)=DRDA*RINV2(I)
  E(I,1,2)=EF(I)/EE1
  E(I,NAP1,2)=EF(I)/EE2
  E(I,1,3)=(DADR-RINV(I)*AMIN1(DELTA1,0.))/EE1
  E(I,NAP1,3)=(DADR-RINV(I)*AMIN1(DELTA2,0.))/EE2
  E(I,1,4)=E(I,1,2)
  E(I,NAP1,4)=E(I,NAP1,2)
  E(I,1,5)=DDRDM/EE1
23 E(I,NAP1,5)=DDRDM/EE2

C      THEN ON INTERIOR OF REGION.

DO 25 I=2,NR
DO 25 J=2,NA
  GAMMA=.5*(PHI(I+1,J,3)-PHI(I-1,J,3))
  DELTA=DA-.5*(PHI(I,J+1,3)-PHI(I,J-1,3))
  E(I,J,6)=EE(I)+RINV(I)*(ABS(GAMMA)+ABS(DELTA))
  E(I,J,1)=(DADR+RINV(I)*AMAX1(DELTA,0.))/E(I,J,6)
  E(I,J,2)=(EF(I)+RINV(I)*AMAX1(GAMMA,0.))/E(I,J,6)
  E(I,J,3)=(DADR-RINV(I)*AMIN1(DELTA,0.))/E(I,J,6)
  E(I,J,4)=(EF(I)-RINV(I)*AMIN1(GAMMA,0.))/E(I,J,6)
25 E(I,J,5)=DDRDM/E(I,J,6)

```

```
C      INITIALIZE SOR ITERATION COUNTER FOR W.

      26 ISOR=0

C      TEST FOR MAXIMUM NUMBER OF SOR ITERATIONS.

      27 IF(ISOR.GE.MAXSOR) GO TO 56

C      UPDATE SOR ITERATION COUNTER IF MAXIMUM HAS NOT BEEN ACHIEVED.

      ISOR=ISOR+1

C      UPDATE PREVIOUS SOR ITERATES FOR W...
C      ...FIRST AT THE ORIGIN

      W(1,1,2)=W(1,1,3)

C      ...THEN ON REMAINDER OF REGION EXCLUDING R = 1.

      DO 29 I=2, NR
      DO 29 J=1, NAP1
      29 W(I,J,2)=W(I,J,3)

C      SET SOR CONVERGENCE INDICATOR (ICONV) TO ZERO.

      ICONV=0

C      COMPUTE CURRENT SOR ITERATE FOR W...
C      ...FIRST AT THE ORIGIN

      W(1,1,3)=RHOC(2)*W(1,1,2)      +RHO(2)*(E1*W(2,1,2)+E2*W(2,NAH,2)
* +E3*W(2,NAP1,2)+E4)

C      (SET VALUES OF W FOR R = 0 AND ALL ANGLES ALPHA(I), I=1,...,NA+1,
C      EQUAL TO VALUE OF W AT ORIGIN FOR CONVENIENCE IN SOR COMPUTATION.)

      DO 30 J=2, NAP1
      30 W(1,J,3)=W(1,1,3)
      IF(ABS(W(1,1,2)-W(1,1,3)).GT.EPPS(2)) ICONV=1

C      ...NEXT ALONG RAY ALPHA = 0. EXCLUDING R = 0 AND R = 1.

      DO 32 I=2, NR
      W(I,1,3)=RHOC(2)*W(I,1,2)+RHO(2)*(E(I,1,1)*W(I+1,1,2)+E(I,1,2)*2.*
* W(I,2,2)+E(I,1,3)*W(I-1,1,3)+E(I,1,5))
      IF(ABS(W(I,1,3)-W(I,1,2)).GT.EPPS(2)) ICONV=1
      32 CONTINUE

C      ...NEXT ON INTERIOR OF REGION

      DO 33 I=2, NR
      DO 33 J=2, NA
      W(I,J,3)=RHOC(2)*W(I,J,2)+RHO(2)*(E(I,J,1)*W(I+1,J,2)+E(I,J,2)*
```

```

* W(I,J+1,2)+E(I,J,3)*W(I-1,J,3)+E(I,J,4)*W(I,J-1,3)+E(I,J,5))
  IF(ABS(W(I,J,2)-W(I,J,3)).GT.EPPS(2)) ICONV=1
33 CONTINUE

C    ...FINALLY ALONG RAY ALPHA = PI. EXCLUDING R = 0 AND R = 1.

      DO 34 I=2,NR
        W(I,NAP1,3)=RHOC(2)*W(I,NAP1,2)+RHO(2)*(E(I,NAP1,1)*W(I+1,NAP1,2)+
* E(I,NAP1,3)*W(I-1,NAP1,3)+2.*E(I,NAP1,4)*W(I,NA,3)+E(I,NAP1,5))
        IF(ABS(W(I,NAP1,2)-W(I,NAP1,3)).GT.EPPS(2)) ICONV=1
34 CONTINUE

C    TEST FOR CONVERGENCE OF SOR ITERATION FOR W.

      IF(ICONV.NE.0) GO TO 27

C    SOR CONVERGENCE ATTAINED FOR W. SMOOTH OUTER ITERATE...
C    ...FIRST AT THE ORIGIN

      W(1,1,3)=XI(2)*W(1,1,1)+XIC(2)*W(1,1,3)

C    (SET VALUES OF W FOR R = 0 AND ALL DISCRETE ANGLES EQUAL TO
C    SMOOTHED VALUE OF W AT ORIGIN.)

      DO 134 J=2,NAP1
134 W(1,J,3)=W(1,1,3)
      IF(ABS(W(1,1,1)-W(1,1,3)).GT.EPS(2)) ICV=1

C    ...THEN ON REMAINDER OF REGION, EXCLUDING R = 1.

      DO 35 I=2,NR
      DO 35 J=I,NAP1
        W(I,J,3)=XI(2)*W(I,J,1)+XIC(2)*W(I,J,3)
        IF(ABS(W(I,J,1)-W(I,J,3)).GT.EPS(2)) ICV=1
35 CONTINUE

C    PRINT OUT SMOOTHED CURRENT OUTER ITERATE FOR W.

      CALL OUTPUT('W',ISOR,W(1,1,3))

C    COMPUTE AND SMOOTH OMEGA ON BOUNDARY R = 1.

      DO 37 J=2,NA
        OMEGA(NRP1,J,3)=XI(3)*OMEGA(NRP1,J,1)-XIC(3)*2.*RINV2(2)*PHI(NR,
* J,3)
        IF(ABS(OMEGA(NRP1,J,1)-OMEGA(NRP1,J,3)).GT.EPS(3)) ICV=1
37 CONTINUE

C    COMPUTE CONSTANT COEFFICIENT FOR SOR ITERATION FOR OMEGA.

      DO 40 I=2,NR
      DO 40 J=2,NA
40 E(I,J,6)=-W(I,J,3)*(SA(J)*(W(I+1,J,3)-W(I-1,J,3)))+CA(I,J)*(W(I,

```

```
* J+1,3)-W(I,J-1,3)))/E(I,J,6)
```

```
C INITIALIZE COUNTER OF SOR ITERATION NUMBER FOR OMEGA.
```

```
41 ISOR=0
```

```
C TEST FOR MAXIMUM NUMBER OF SOR ITERATIONS.
```

```
42 IF(ISOR.GE.MAXSOR) GO TO 57
```

```
C UPDATE SOR ITERATION COUNTER IF MAXIMUM NUMBER HAS NOT BEEN  
C ACHIEVED.
```

```
ISOR=ISOR+1
```

```
C UPDATE PREVIOUS SOR ITERATES FOR OMEGA ON INTERIOR.
```

```
DO 44 I=2, NR
```

```
DO 44 J=2, NA
```

```
44 OMEGA(I, J, 2)=OMEGA(I, J, 3)
```

```
C SET SOR CONVERGENCE INDICATOR TO ZERO.
```

```
ICONV=0
```

```
C COMPUTE CURRENT SOR ITERATE FOR OMEGA ON INTERIOR.
```

```
DO 46 I=2, NR
```

```
DO 46 J=2, NA
```

```
OMEGA(I, J, 3)=RHOC(3)*OMEGA(I, J, 2)+RHO(3)*(E(I, J, 1)*OMEGA(I+1, J, 2)
```

```
* +E(I, J, 2)*OMEGA(I, J+1, 2)+E(I, J, 3)*OMEGA(I-1, J, 3)+E(I, J, 4)
```

```
* *OMEGA(I, J-1, 3)+E(I, J, 6))
```

```
IF(ABS(OMEGA(I, J, 2)-OMEGA(I, J, 3)).GT.EPPS(3)) ICONV=1
```

```
46 CONTINUE
```

```
C TEST FOR CONVERGENCE OF SOR ITERATION FOR OMEGA.
```

```
IF(ICONV.NE.0) GO TO 42
```

```
C SOR CONVERGENCE ATTAINED FOR OMEGA. SMOOTH OUTER ITERATE.
```

```
DO 48 I=2, NR
```

```
DO 48 J=2, NA
```

```
OMEGA(I, J, 3)=XI(4)*OMEGA(I, J, 1)+XIC(4)*OMEGA(I, J, 3)
```

```
IF(ABS(OMEGA(I, J, 1)-OMEGA(I, J, 3)).GT.EPS(3)) ICV=1
```

```
48 CONTINUE
```

```
C PRINT OUT SMOOTHED CURRENT OUTER ITERATE FOR OMEGA.
```

```
CALL OUTPUT('OMEGA', ISOR, OMEGA(1, 1, 3))
```

```
C TEST FOR CONVERGENCE OF ALL OUTER ITERATES.
```

```
IF(ICV.NE.0) GO TO 10

C PRINT MESSAGE INDICATING CONVERGENCE OF OUTER ITERATION.

WRITE(6,97)

C PUNCH SOLUTION (AND VALUE OF  $\theta$ ) OUT ON CARDS IF INPUT CONTROL
C PARAMETER IPCH SO DICTATES, AND PRINT MESSAGE INDICATING THAT
C THIS PUNCHING WAS DONE.

IF(IPCH.NE.0)PUNCH88,D,((PHI(I,J,3),J=1,NAP1),I=1,NRP1),((W(I,J,3)
* ,J=1,NAP1),I=1,NRP1),((OMEGA(I,J,3),J=1,NAP1),I=1,NRP1)
IF(IPCH.NE.0) WRITE(6,86)

C COMPUTE RATIO OF FLUX IN CURVED TUBE TO THAT IN STRAIGHT TUBE.

QR=0.
DO 49 J=1,NA
49 QR=QR+W(1,J,3)+W(2,J,3)+W(2,J+1,3)
QR=QR*CO(1)
DO 249 I=2,NR
S=0.
DO 149 J=1,NA
149 S=S+W(I,J,3)+W(I+1,J,3)+W(I,J+1,3)+W(I+1,J+1,3)
249 QR=QR+CO(I)*S

C PRINT OUT VALUE OF FLUX RATIO JUST COMPUTED.

WRITE(6,90) QR

C TEST WHETHER OR NOT SOLUTION JUST OBTAINED IS TO BE SAVED IN
C MEMORY FOR SOME SUCCEEDING CASE WITH A DIFFERENT VALUE OF D.

IF(ISAVE.EQ.0) GO TO 5

C SAVE CURRENT SOLUTION IN ANOTHER AREA OF MEMORY.

DO 50 I=1,NRP1
DO 50 J=1,NAP1
PHI(I,J,4)=PHI(I,J,3)
W(I,J,4)=W(I,J,3)
50 OMEGA(I,J,4)=OMEGA(I,J,3)

C SAVE VALUE OF D FOR WHICH SOLUTION WAS JUST OBTAINED IN DSTART.

DSTART=D
GO TO 5

C CONVERGENCE FAILURE MESSAGES

C PHI ITERATION FAILED. READ NEXT DATA CASE.

55 WRITE(6,96)
```

```
GO TO 5
```

```
C W ITERATION FAILED. REDUCE OVER-RELAXATION FACTOR AND TRY AGAIN,  
C UNLESS THIS HAS ALREADY BEEN DONE THE MAXIMUM NUMBER OF TIMES  
C (NOR), IN WHICH CASE READ NEXT DATA CASE.
```

```
56 WRITE(6,95) RH0(2)  
IF(IRW.GE.NOR) GO TO 5  
IRW=IRW+1  
RH0(2)=RH0(2)-DOR(IRW)  
RHOC(2)=1.-RH0(2)  
DO 156 I=1,NRP1  
DO 156 J=1,NAP1  
156 W(I,J,3)=W(I,J,1)  
GO TO 28
```

```
C OMEGA ITERATION FAILED. REDUCE O-R FACTOR AND TRY AGAIN, UNLESS  
C THIS HAS ALREADY BEEN DONE THE MAXIMUM NUMBER OF TIMES (NOR),  
C IN WHICH CASE READ THE NEXT DATA CASE.
```

```
57 WRITE(6,94) RH0(3)  
IF(IRO.GE.NOR) GO TO 5  
IRO=IRO+1  
RH0(3)=RH0(3)-DOR(IRO)  
RHOC(3)=1.-RH0(3)  
DO 157 I=1,NRP1  
DO 157 J=1,NAP1  
157 OMEGA(I,J,3)=OMEGA(I,J,1)  
GO TO 41
```

```
C OUTER ITERATION FAILED. PRINT MESSAGE INDICATING THIS AND READ  
C NEXT DATA CASE.
```

```
58 WRITE(6,93)  
GO TO 5
```

```
C TERMINATION POINT FOR PROGRAM. CONTROL REACHES HERE AFTER ATTEMPT  
C TO READ PAST LAST DATA RECORD.
```

```
70 STOP
```

```
99 FORMAT(11F5.5,3I5)  
98 FORMAT(1H1 6X 'D =' F5.0//13X 'PHI' 7X 'W' 5X 'OMEGA' // 5X  
* 'RHO =' F6.2,2(3X F6.2)/6X 'XI =' F6.4,3(3X F6.4)/5X 'EPS ='  
* F6.4,2(3X F6.4))  
97 FORMAT('O OUTER ITERATION CONVERGED TO GIVEN TOLERANCES.')
```

```
96 FORMAT('OSOR FOR PHI FAILED.')
```

```
95 FORMAT('OSOR FOR W FAILED WITH SOR FACTOR =' F6.2)
```

```
94 FORMAT('OSOR FOR OMEGA FAILED WITH SOR FACTOR =' F6.2)
```

```
93 FORMAT('O OUTER ITERATION FAILED TO CONVERGE.')
```

```
92 FORMAT('//O OUTER ITERATION' I5//)
```

```
91 FORMAT(1X 11E11.5)
```

```
90 FORMAT('O FLUX RATIO =' F10.5)
```

```
89 FORMAT('OSOR FACTOR FOR W CHANGED TO ' F6.2)
88 FORMAT(8E10.5)
87 FORMAT('OINITIAL ITERATE TAKEN FROM SOLUTION FOR D =' F7.0)
86 FORMAT('OSOLUTION WAS OUTPUT ON PUNCHED CARDS.')
```

END

C THIS ROUTINE PRINTS OUT A DISCRETE FUNCTION IN A RECTANGULAR
C FORMAT WHICH IS RELATED TN THE FOLLOWING WAY TO THE POLAR GRID
C IMPOSED ON THE PHYSICAL REGION.
C THE TOP LINE OF THE PRINT BLOCK CORRESPONDS TO VALUES OF THE
C FUNCTION ALONG THE ARC $R = 1$.
C THE BOTTOM LINE OF THE PRINT BLOCK CORRESPONDS TO THE VALUE
C OF THE FUNCTION AT THE ORIGIN ($R = 0$).

```
SUBROUTINE OUTPUT(VAR, ISOR, A)
PARAMETER NR=10, NA=18, NRP1=NR+1, NAP1=NA+1
DIMENSION A(NRP1, NAP1)
DATA APMI, AW, AOMEGA/'PHI ', 'W ', 'OMEGA '/
99 FORMAT(1H0 A6, 5X I5, 2X 'SOR ITERATIONS'/)
98 FORMAT(1X F6.2, 17F7.2, F6.2)
97 FORMAT(1X F6.1, 17F7.1, F6.1)
KRP1=NRP1
KAP1=NAP1
WRITE(6, 99) VAR, ISOR
IF(VAR.NE.APMI) GO TO 8
DO 5 I=1, NRP1
5 WRITE(6, 98) (A(KRP1-I+1, KAP1-J+1), J=1, KAP1)
RETURN
8 DO 10 I=1, NRP1
10 WRITE(6, 97) (A(KRP1-I+1, KAP1-J+1), J=1, KAP1)
RETURN
END
```

Appendix B. Minimal modern Fortran code

This appendix includes the minimally modernised Fortran version of the original FORTRAN 66 code written by A.B.Schubert in 1972 [4].

Compiling and running the solution creates a separate file for each value of D .

```
C   Appendix: FORTRAN Program for Secondary Flow, by A. B. Schubert (1972)
C
C   Nils T. Basse (2025):
C       Updates to remove obsolescent features (Hollerith strings and
C       data) and to write results to files instead of to punch cards.
C   Other changes marked by "NTB 2025" below.
C
C   TUBEFLOW   A PROGRAM WHICH COMPUTES THE SECONDARY
C   (CROSS-SECTIONAL) FLOW OF A FLUID IN A CURVED TUBE AND THE
C   VELOCITY AND FLUX IN THE AXIAL DIRECTION OF THE TUBE BY
C   DISCRETIZATION OF THE CROSS-SECTION AND OF THE GOVERNING
C   DIFFERENTIAL EQUATIONS.
C
C   NTB 2025:
C   Original (coarse) resolution: NR=10,NA=18
C   Updated (fine) resolution:   NR=20,NA=36
C   INTEGER, PARAMETER :: NR = 10
C   INTEGER, PARAMETER :: NA = 18
C
C   PARAMETER NRP1=NR+1,NAP1=NA+1,NRM1=NR-1,NAM1=NA-1,
C   * NAH=NA/2+1
C
C   NR = NUMBER OF GRID SPACES ON (0,1) IN RADIAL (R) DIRECTION.
C   NA = NUMBER OF GRID SPACES ON (0,PI) IN ANGULAR (ALPHA)
C   DIRECTION.
C
C   PRINCIPAL DISCRETE FUNCTIONS COMPUTED ARE DEFINED AS FOLLOWS.
C   PHI(I,J,K) = NON-DIMENSIONAL STREAM FUNCTION VALUE FOR SECONDARY
C   FLOW AT POLAR GRID POINT CORRESPONDING TO I-TH
C   RADIAL VALUE (WHERE I= 1 CORRESPONDS TO R = 0 AND
C   I = NR+1 CORRESPONDS TO R = 1) AND J-TH ANGULAR
C   VALUE (WHERE J = 1 CORRESPONDS TO ALPHA = 0 AND
C   J = NA+1 CORRESPONDS TO ALPHA = PI.
C   K = 1 IMPLIES A VALUE AT THE PREVIOUS OUTER
C   ITERATION.
C   K = 2 IMPLIES A VALUE AT THE PREVIOUS SOR ITERATION
C   K = 3 IMPLIES A VALUE AT THE CURRENT SOR AND OUTER
C   ITERATION.
C   W(I,J,K) = NON-DIMENSIONAL VELOCITY IN THE AXIAL DIRECTION,
C   WITH SUBSCRIPTS DEFINED AS FOR PHI.
C   OMEGA(I,J,K) = INTERMEDIATE VARIABLE INTRODUCED IN ANALYTICAL
C   DESCRIPTION OF PROBLEM, WITH SUBSCRIPTS DEFINED
C   AS FOR PHI.
C
C   DIMENSION PHI(NRP1,NAP1,4),W(NRP1,NAP1,4),OMEGA(NRP1,NAP1,4),
C   * XI(4),RHO(3),EPS(3),SA(NAP1),COSA(NAP1),RINV(NRP1),RINV2(NRP1),
C   * RHOC(3),B(NRP1,5),C(NRP1,NAP1),EPPS(3),XIC(4),E(NRP1,NAP1,6),
C   * EE(NRP1),EF(NRP1),CA(NRP1,NAP1),DELA(NR),CO(NR),DOR(3)
C
C   NTB 2025:
C   MAXSOR and MAXOUT increased by a factor of ten for fine resolution
C   DATA PI/3.14159255/,MAXSOR,MAXOUT/2500,600/,
C   * NOR/3/, (DOR(I),I=1,3)
```

```
*/.2,.2,.2/
```

```
C   NTB 2025:
C   Added definitions to write files
CHARACTER (LEN=10) :: file_id
CHARACTER (LEN=50) :: file_name
INTEGER :: ctr

C   MAXSOR = MAXIMUM NUMBER OF ITERATIONS TO BE ALLOWED FOR COMPUTING
C   SOLUTION TO ANY SYSTEM BY SUCCESSIVE OVER-RELAXATION
C   MAXOUT = MAXIMUM NUMBER OF OUTER ITERATIONS TO BE ALLOWED IN THE
C   COMPUTATION OF PHI, W, AND OMEGA.
C   NOR = NUMBER OF DIFFERENT OVER-RELAXATION FACTORS TO BE TRIED
C   IN ANY DATA CASE.
C   DOR(I) = AMOUNT OF DECREASE IN OVER-RELAXATION FACTOR AT I-TH
C   CHANGE.

KRP1=NRP1

C   DR = GRID SPACING IN RADIAL DIRECTION.
C   DA = GRID SPACING IN ANGULAR DIRECTION.

DR=1./NR
DA=PI/NA
DAH=.5*DA
DRH=.5*DR
DRDAM=DR*DA
DRDA=DR/DA
DADR=DA/DR

C   COMPUTE ELEMENTS OF AREA, DELA(I), I=1,...,NR, IN RADIAL DIRECTION
C   FOR USE IN FLUX CALCULATION.

DO 1 I=1,NR
1 DELA(I)=(2*I-1)*DRH*DRDAM

C   ON THE GRID DEFINED BY DR AND DA, COMPUTE DISCRETE FUNCTIONS
C   DEPENDENT ONLY ON RADIUS AND ANGLE.

SA(1)=0.
DO 2 J=2,NA
SA(J)=SIN((J-1)*DA)*DAH
2 COSA(J)=COS((J-1)*DA)
SA(NAP1)=0.
COSA(NAP1)=-1.
RINV(NRP1)=1.
RINV2(NRP1)=1.
DO 3 I=2,NR
RINV(I)=1./((I-1)*DR)
DRRH=DRH*RINV(I)
RINV2(I)=RINV(I)**2
DO 3 J=2,NA
3 CA(I,J)=DRRH*COSA(J)
```

```
C    COMPUTE SOR COEFFICIENTS FOR PHI.

      DO 4 I=2,NR
      BO=2.*(DADR+DRDA*RINV2(I))+DA*RINV(I)
      B(I,1)=(DADR+DA*RINV(I))/BO
      B(I,2)=DRDA*RINV2(I)/BO
      B(I,3)=DADR/BO
      B(I,4)=B(I,2)
4     B(I,5)=DRDAM/BO

C    READ OUTER ITERATION SMOOTHING PARAMETERS (XI), OVER-RELAXATION
C    FACTORS (RHO), OUTER ITERATION CONVERGENCE TOLERANCES (EPS),
C    PARAMETER RELATED TO REYNOLDS NO. (D), AND INPUT/OUTPUT CONTROL
C    VARIABLES DEFINED AS FOLLOWS.
C    ISAVE.NE.0 IMPLIES SAVE THE SOLUTION FOR THIS CASE (IF OBTAINED)
C           IN MEMORY FOR USE AS STARTING VALUES FOR A SUCCEEDING
C           CASE WITH A DIFFERENT VALUE OF D.
C    ISAVE.EQ.0 IMPLIES DO NOT DO THE ABOVE.
C    IUSE.NE.0 IMPLIES TAKE STARTING VALUES FOR OUTER ITERATION FROM
C           MEMORY.
C    IUSE.EQ.0 IMPLIES INITIALIZE OUTER ITERATES TO ZERO.
C    IPCH.NE.0 IMPLIES WRITE SOLUTION FOR THIS CASE (IF OBTAINED)
C           TO FILE
C    IPCH.EQ.0 IMPLIES DO NOT WRITE SOLUTION OBTAINED FOR THIS CASE
C           TO FILE.

C    NTB 2025:
C    Save solution to memory
C    Take starting values from memory
C    Write solution to file
      ISAVE=1
      IUSE=1
      IPCH=1

C    NTB 2025:
C    Set initial conditions for all cases treated
      ctr=0
5     ctr=ctr+1
      IF (ctr==1) THEN
        D=10
        EPS(1)=10.**(-5.)
        EPS(2)=10.**(-3.)
        EPS(3)=10.**(-4.)
        RHO(1)=1.5
        RHO(2)=1.8
        RHO(3)=1.5
        XI(1)=0.1
        XI(2)=0.1
        XI(3)=0.1
        XI(4)=0.1
      ELSE IF (ctr==2) THEN
        D=100
```

```
      EPS(1)=2.*10.**(-4.)
      EPS(2)=5.*10.**(-3.)
      EPS(3)=5.*10.**(-3.)
      RHO(1)=1.5
      RHO(2)=1.7
      RHO(3)=1.5
      XI(1)=0.1
      XI(2)=0.1
      XI(3)=0.1
      XI(4)=0.1
      ELSE IF (ctr==3) THEN
        D=250
        EPS(1)=2.*10.**(-3.)
        EPS(2)=2.*10.**(-2.)
        EPS(3)=4.*10.**(-2.)
        RHO(1)=1.5
C      NTB 2025:
C      RHO(2) changed from 1.8 (iteration 1), then 1.5
C      to 1.5 for all iterations
        RHO(2)=1.5
        RHO(3)=1.5
        XI(1)=0.1
        XI(2)=0.1
        XI(3)=0.1
        XI(4)=0.1
      ELSE IF (ctr==4) THEN
        D=500
        EPS(1)=4.*10.**(-3.)
        EPS(2)=4.*10.**(-2.)
        EPS(3)=8.*10.**(-2.)
        RHO(1)=1.5
        RHO(2)=1.5
        RHO(3)=1.5
        XI(1)=0.1
        XI(2)=0.1
        XI(3)=0.1
        XI(4)=0.1
      ELSE IF (ctr==5) THEN
        D=1000
        EPS(1)=5.*10.**(-3.)
        EPS(2)=5.*10.**(-2.)
        EPS(3)=17.*10.**(-2.)
        RHO(1)=1.5
        RHO(2)=1.5
        RHO(3)=1.5
        XI(1)=0.5
        XI(2)=0.1
        XI(3)=0.1
        XI(4)=0.5
      ELSE IF (ctr==6) THEN
        D=2000
        EPS(1)=7.*10.**(-3.)
        EPS(2)=8.*10.**(-2.)
```

```
      EPS(3)=3.*10.**(-1.)
C      NTB 2025:
C      RHO(1), RHO(2) and RHO(3) multiplied by a factor 0.8
      RHO(1)=0.8*1.5
      RHO(2)=0.8*1.5
      RHO(3)=0.8*1.3
      XI(1)=0.5
      XI(2)=0.1
      XI(3)=0.1
      XI(4)=0.5
      ELSE IF (ctr==7) THEN
        D=5000
        EPS(1)=10.**(-2.)
        EPS(2)=15.*10.**(-2.)
        EPS(3)=6.*10.**(-1.)
C      NTB 2025:
C      RHO(1), RHO(2) and RHO(3) multiplied by a factor 0.8
      RHO(1)=0.8*1.5
      RHO(2)=0.8*1.5
      RHO(3)=0.8*1.3
      XI(1)=0.3
      XI(2)=0.1
      XI(3)=0.1
      XI(4)=0.7
      ELSE
        PRINT *, 'D-loop complete: To exit, press <ENTER>'
        READ(*,*)
        STOP
      END IF

C      PRINT OUT INPUT PARAMETERS.

      WRITE(6,98) D,RHO,XI,EPS

C      COMPUTE PARAMETERS DEPENDENT ON THE INPUT PARAMETER D, INCLUDING
C      THE COEFFICIENTS, CO(I),I=1,...,NR, IN THE NUMERICAL INTEGRATION
C      FOR THE FLUX.

      DDRDAM=D*DRDAM
      DDR2=D*DR**2
      CON=4./(PI*D)
      CO(1)=16.*DELA(1)/(3.*PI*D)
      DO 105 I=2,NR
105    CO(I)=CON*DELA(I)

C      COMPUTE COMPLEMENTS (RELATIVE TO 1) OF SMOOTHING PARAMETERS AND
C      OVER-RELAXATION FACTORS.
C      ALSO COMPUTE SOR CONVERGENCE TOLERANCES, EPPS(I),I=1,2,3, AS
C      FUNCTIONS OF OUTER ITERATION TOLERANCES, EPS(I),I=1,2,3.

      DO 6 I=1,3
      XIC(I)=1.-XI(I)
      RHOC(I)=1.-RHO(I)
```

```
6 EPPS(I)=.05*EPS(I)
  XIC(4)=1.-XI(4)

C   TEST WHETHER INITIAL OUTER ITERATES ARE TO BE OBTAINED FROM
C   MEMORY, HAVING BEEN PLACED THERE AS THE SOLUTION FOR A PREVIOUS
C   VALUE OF D BY A PREVIOUS COMPUTATION THIS RUN.

      IF(IUSE.EQ.0) GO TO 7
      DO 106 I=1,NRP1
      DO 106 J=1,NAP1
      PHI(I,J,3)=PHI(I,J,4)
      W(I,J,3)=W(I,J,4)
106  OMEGA(I,J,3)=OMEGA(I,J,4)
      WRITE(6,87) DSTART
      GO TO 9

C   INITIALIZE OUTER ITERATES TO ZERO IF NEITHER INPUT NOR COMPUTED
C   PREVIOUSLY THIS RUN.

7 DO 8 I=1,NRP1
  DO 8 J=1,NAP1
  PHI(I,J,3)=0.
  W(I,J,3)=0.
8  OMEGA(I,J,3)=0.

C   INITIALIZE OUTER ITERATION COUNTER (IOUT) AND COUNTERS OF NUMBER
C   OF OVER-RELAXATION FACTORS USED FOR W AND OMEGA (IRW,IRO, RESP.)
C   FOR THIS DATA CASE.

9 IOUT=0
  IRW=0
  IRO=0

C   TEST FOR MAXIMUM NUMBER OF OUTER ITERATIONS.

10 IF(IOUT.GE.MAXOUT) GO TO 58

C   UPDATE OUTER ITERATION COUNTER, WRITE MESSAGE INDICATING NEW
C   OUTER ITERATION NUMBER, AND SET OUTER ITERATION CONVERGENCE
C   INDICATOR (ICV) TO ZERO. AFTER COMPUTATION OF ALL CURRENT OUTER
C   ITERATES AND COMPARISON WITH PREVIOUS OUTER ITERATES AGAINST THE
C   SPECIFIED TOLERANCES, ICV WILL STILL BE ZERO IF CONVERGENCE HAS
C   BEEN OBTAINED, OTHERWISE ICV WILL BE NON-ZERO.

      IOUT=IOUT+1
      WRITE(6,92) IOUT
      ICV=0

C   UPDATE PREVIOUS OUTER ITERATES.
      DO 12 I=1,NRP1
      DO 12 J=1,NAP1
      PHI(I,J,1)=PHI(I,J,3)
      W(I,J,1)=W(I,J,3)
```

```
12 OMEGA(I,J,1)=OMEGA(I,J,3)

C   COMPUTE CONSTANT SOR COEFFICIENT FOR PHI.

      DO 13 I=2,NR
      DO 13 J=2,NA
13  C(I,J)=B(I,5)*OMEGA(I,J,1)

C   INITIALIZE SOR ITERATION COUNTER FOR PHI.

      ISOR=0

C   TEST FOR MAXIMUM NUMBER OF SOR ITERATIONS FOR PHI.

14  IF(ISOR.GE.MAXSOR) GO TO 55

C   UPDATE SOR ITERATION COUNTER IF MAXIMUM NUMBER HAS NOT BEEN
C   ACHIEVED.

      ISOR=ISOR+1

C   UPDATE PREVIOUS SOR ITERATES FOR PHI.

      DO 16 I=2,NR
      DO 16 J=2,NA
16  PHI(I,J,2)=PHI(I,J,3)

C   SET SOR ITERATION CONVERGENCE INDICATOR (ICONV) TO ZERO. AFTER
C   COMPUTATION OF CURRENT SOR ITERATE AND COMPARISON WITH PREVIOUS
C   ITERATE AGAINST THE TOLERANCE, ICONV WILL STILL BE ZERO IF
C   CONVERGENCE HAS BEEN OBTAINED. OTHERWISE IT WILL BE NON-ZERO.

      ICONV=0

C   COMPUTE CURRENT SOR ITERATE FOR PHI...
C   ...FIRST ON INTERIOR OF REGION, EXCLUDING 'INNER BOUNDARY'
C   R = 1.-DR.

      DO 18 I=2,NRM1
      DO 18 J=2,NA
          PHI(I,J,3)=RHOC(1)*PHI(I,J,2)+RHO(1)*(B(I,1)*PHI(I+1,J,2)
* +B(I,2)*PHI(I,J+1,2)+B(I,3)*PHI(I-1,J,3)+B(I,4)*PHI(I,J-1,3)
* +C(I,J))
          IF(ABS(PHI(I,J,2)-PHI(I,J,3)).GT.EPPS(1)) ICONV=1
18  CONTINUE

C   ...THEN ON 'INNER BOUNDARY' R = 1.-DR.

      DO 19 J=2,NA
          PHI(NR,J,3)=RHOC(1)*PHI(NR,J,2)+RHO(1)*.25*PHI(NRM1,J,3)
          IF(ABS(PHI(NR,J,2)-PHI(NR,J,3)).GT.EPPS(1)) ICONV=1
19  CONTINUE
```

```

C     TEST FOR CONVERGENCE OF SOR ITERATION FOR PHI.

      IF(ICONV.NE.0) GO TO 14

C     SOR CONVERGENCE ATTAINED FOR PHI. SMOOTH OUTER ITERATE.
DO 20 I=2, NR
DO 20 J=2, NA
      PHI(I, J, 3)=XI(1)*PHI(I, J, 1)+XIC(1)*PHI(I, J, 3)
      IF(ABS(PHI(I, J, 1)-PHI(I, J, 3)).GT.EPS(1)) ICV=1
20 CONTINUE

C     PRINT OUT SMOOTHED CURRENT OUTER ITERATE FOR PHI.

C     NTB 2025:
C     Added NR and NA as inputs
CALL OUTPUT('PHI ', ISOR, PHI(1, 1, 3), NR, NA)

C     COMPUTE COEFFICIENTS FOR SOR ITERATION FOR W...
C     ...FIRST COMPUTE COEFFICIENTS AT THE ORIGIN

E0=4.+ABS(PHI(2, NAH, 3))
E1=(1.-AMIN1(PHI(2, NAH, 3), 0.))/E0
E2=2./E0
E3=(1.+AMAX1(PHI(2, NAH, 3), 0.))/E0
E4=DDR2/E0

C     ...NEXT COMPUTE COEFFICIENTS FOR REMAINDER OF REGION...
C     FIRST FOR RAYS ALONG ALPHA = 0 AND ALPHA = PI, EXCLUDING
C     R = 0 AND R = 1.

DO 23 I=2, NR
DELTA1=DA-PHI(I, 2, 3)
DELTA2=DA+PHI(I, NA, 3)
EE(I)=2.*(DADR+DRDA*RINV2(I))
EE1=EE(I)+RINV(I)*ABS(DELTA1)
EE2=EE(I)+RINV(I)*ABS(DELTA2)
E(I, 1, 1)=(DADR+RINV(I)*AMAX1(DELTA1, 0.))/EE1
E(I, NAP1, 1)=(DADR+RINV(I)*AMAX1(DELTA2, 0.))/EE2
EF(I)=DRDA*RINV2(I)
E(I, 1, 2)=EF(I)/EE1
E(I, NAP1, 2)=EF(I)/EE2
E(I, 1, 3)=(DADR-RINV(I)*AMIN1(DELTA1, 0.))/EE1
E(I, NAP1, 3)=(DADR-RINV(I)*AMIN1(DELTA2, 0.))/EE2
E(I, 1, 4)=E(I, 1, 2)
E(I, NAP1, 4)=E(I, NAP1, 2)
E(I, 1, 5)=DDRDM/EE1
23 E(I, NAP1, 5)=DDRDM/EE2

C     THEN ON INTERIOR OF REGION.

DO 25 I=2, NR
DO 25 J=2, NA
GAMMA=.5*(PHI(I+1, J, 3)-PHI(I-1, J, 3))

```

```

      DELTA=DA-.5*(PHI(I,J+1,3)-PHI(I,J-1,3))
      E(I,J,6)=EE(I)+RINV(I)*(ABS(GAMMA)+ABS(DELTA))
      E(I,J,1)=(DADR+RINV(I)*AMAX1(DELTA,0.))/E(I,J,6)
      E(I,J,2)=(EF(I)+RINV(I)*AMAX1(GAMMA,0.))/E(I,J,6)
      E(I,J,3)=(DADR-RINV(I)*AMIN1(DELTA,0.))/E(I,J,6)
      E(I,J,4)=(EF(I)-RINV(I)*AMIN1(GAMMA,0.))/E(I,J,6)
25  E(I,J,5)=DDRDAM/E(I,J,6)

C      INITIALIZE SOR ITERATION COUNTER FOR W.

26  ISOR=0

C      TEST FOR MAXIMUM NUMBER OF SOR ITERATIONS.

27  IF(ISOR.GE.MAXSOR) GO TO 56

C      UPDATE SOR ITERATION COUNTER IF MAXIMUM HAS NOT BEEN ACHIEVED.

      ISOR=ISOR+1

C      UPDATE PREVIOUS SOR ITERATES FOR W...
C      ...FIRST AT THE ORIGIN

      W(1,1,2)=W(1,1,3)

C      ...THEN ON REMAINDER OF REGION EXCLUDING R = 1.

      DO 29 I=2,NR
      DO 29 J=1,NAP1
29  W(I,J,2)=W(I,J,3)

C      SET SOR CONVERGENCE INDICATOR (ICONV) TO ZERO.

      ICONV=0

C      COMPUTE CURRENT SOR ITERATE FOR W...
C      ...FIRST AT THE ORIGIN

      W(1,1,3)=RHOC(2)*W(1,1,2)+RHO(2)*(E1*W(2,1,2)+E2*W(2,NAH,2)
      +E3*W(2,NAP1,2)+E4)

C      (SET VALUES OF W FOR R = 0 AND ALL ANGLES ALPHA(I), I=1,...,NA+1,
C      EQUAL TO VALUE OF W AT ORIGIN FOR CONVENIENCE IN SOR COMPUTATION.)

      DO 30 J=2,NAP1
30  W(1,J,3)=W(1,1,3)
      IF(ABS(W(1,1,2)-W(1,1,3)).GT.EPPS(2)) ICONV=1

C      ...NEXT ALONG RAY ALPHA = 0. EXCLUDING R = 0 AND R = 1.

      DO 32 I=2,NR
      W(I,1,3)=RHOC(2)*W(I,1,2)+RHO(2)*(E(I,1,1)*W(I+1,1,2)+E(I,1,2)*2.*
      W(I,2,2)+E(I,1,3)*W(I-1,1,3)+E(I,1,5))

```

```

      IF(ABS(W(I,1,3)-W(I,1,2)).GT.EPPS(2)) ICONV=1
32 CONTINUE

C     ...NEXT ON INTERIOR OF REGION

      DO 33 I=2, NR
      DO 33 J=2, NA
      W(I, J, 3)=RHOC(2)*W(I, J, 2)+RHO(2)*(E(I, J, 1)*W(I+1, J, 2)+E(I, J, 2)*
* W(I, J+1, 2)+E(I, J, 3)*W(I-1, J, 3)+E(I, J, 4)*W(I, J-1, 3)+E(I, J, 5))
      IF(ABS(W(I, J, 2)-W(I, J, 3)).GT.EPPS(2)) ICONV=1
33 CONTINUE

C     ...FINALLY ALONG RAY ALPHA = PI. EXCLUDING R = 0 AND R = 1.

      DO 34 I=2, NR
      W(I, NAP1, 3)=RHOC(2)*W(I, NAP1, 2)+RHO(2)*(E(I, NAP1, 1)*W(I+1, NAP1, 2)+
* E(I, NAP1, 3)*W(I-1, NAP1, 3)+2.*E(I, NAP1, 4)*W(I, NA, 3)+E(I, NAP1, 5))
      IF(ABS(W(I, NAP1, 2)-W(I, NAP1, 3)).GT.EPPS(2)) ICONV=1
34 CONTINUE

C     TEST FOR CONVERGENCE OF SOR ITERATION FOR W.

      IF(ICONV.NE.0) GO TO 27

C     SOR CONVERGENCE ATTAINED FOR W. SMOOTH OUTER ITERATE...
C     ...FIRST AT THE ORIGIN

      W(1, 1, 3)=XI(2)*W(1, 1, 1)+XIC(2)*W(1, 1, 3)

C     (SET VALUES OF W FOR R = 0 AND ALL DISCRETE ANGLES EQUAL TO
C     SMOOTHED VALUE OF W AT ORIGIN.)

      DO 134 J=2, NAP1
134 W(1, J, 3)=W(1, 1, 3)
      IF(ABS(W(1, 1, 1)-W(1, 1, 3)).GT.EPS(2)) ICV=1

C     ...THEN ON REMAINDER OF REGION, EXCLUDING R = 1.

      DO 35 I=2, NR
      DO 35 J=I, NAP1
      W(I, J, 3)=XI(2)*W(I, J, 1)+XIC(2)*W(I, J, 3)
      IF(ABS(W(I, J, 1)-W(I, J, 3)).GT.EPS(2)) ICV=1
35 CONTINUE

C     PRINT OUT SMOOTHED CURRENT OUTER ITERATE FOR W.

C     NTB 2025:
C     Added NR and NA as inputs
      CALL OUTPUT('W ', ISOR, W(1, 1, 3), NR, NA)

C     COMPUTE AND SMOOTH OMEGA ON BOUNDARY R = 1.

      DO 37 J=2, NA

```

```
      OMEGA(NRP1,J,3)=XI(3)*OMEGA(NRP1,J,1)-XIC(3)*2.*RINV2(2)*PHI(NR,  
* J,3)  
      IF(ABS(OMEGA(NRP1,J,1)-OMEGA(NRP1,J,3)).GT.EPS(3)) ICV=1  
37 CONTINUE  
  
C      COMPUTE CONSTANT COEFFICIENT FOR SOR ITERATION FOR OMEGA.  
  
      DO 40 I=2, NR  
      DO 40 J=2, NA  
40 E(I,J,6)=-W(I,J,3)*(SA(J)*(W(I+1,J,3)-W(I-1,J,3))+CA(I,J)*(W(I,  
* J+1,3)-W(I,J-1,3)))/E(I,J,6)  
  
C      INITIALIZE COUNTER OF SOR ITERATION NUMBER FOR OMEGA.  
  
41 ISOR=0  
  
C      TEST FOR MAXIMUM NUMBER OF SOR ITERATIONS.  
  
42 IF(ISOR.GE.MAXSOR) GO TO 57  
  
C      UPDATE SOR ITERATION COUNTER IF MAXIMUM NUMBER HAS NOT BEEN  
C      ACHIEVED.  
  
      ISOR=ISOR+1  
  
C      UPDATE PREVIOUS SOR ITERATES FOR OMEGA ON INTERIOR.  
  
      DO 44 I=2, NR  
      DO 44 J=2, NA  
44 OMEGA(I,J,2)=OMEGA(I,J,3)  
  
C      SET SOR CONVERGENCE INDICATOR TO ZERO.  
  
      ICONV=0  
  
C      COMPUTE CURRENT SOR ITERATE FOR OMEGA ON INTERIOR.  
  
      DO 46 I=2, NR  
      DO 46 J=2, NA  
      OMEGA(I,J,3)=RHOC(3)*OMEGA(I,J,2)+RHO(3)*(E(I,J,1)*OMEGA(I+1,J,2)  
* +E(I,J,2)*OMEGA(I,J+1,2)+E(I,J,3)*OMEGA(I-1,J,3)+E(I,J,4)  
* *OMEGA(I,J-1,3)+E(I,J,6))  
      IF(ABS(OMEGA(I,J,2)-OMEGA(I,J,3)).GT.EPPS(3)) ICONV=1  
46 CONTINUE  
  
C      TEST FOR CONVERGENCE OF SOR ITERATION FOR OMEGA.  
  
      IF(ICONV.NE.0) GO TO 42  
  
C      SOR CONVERGENCE ATTAINED FOR OMEGA. SMOOTH OUTER ITERATE.  
  
      DO 48 I=2, NR  
      DO 48 J=2, NA
```

```
      OMEGA(I,J,3)=XI(4)*OMEGA(I,J,1)+XIC(4)*OMEGA(I,J,3)
      IF(ABS(OMEGA(I,J,1)-OMEGA(I,J,3)).GT.EPS(3)) ICV=1
48  CONTINUE

C     PRINT OUT SMOOTHED CURRENT OUTER ITERATE FOR OMEGA.

C     NTB 2025:
C     Added NR and NA as inputs
      CALL OUTPUT('OMEGA',ISOR,OMEGA(1,1,3),NR,NA)

C     TEST FOR CONVERGENCE OF ALL OUTER ITERATES.

      IF(ICV.NE.0) GO TO 10

C     PRINT MESSAGE INDICATING CONVERGENCE OF OUTER ITERATION.

      WRITE(6,97)

C     COMPUTE RATIO OF FLUX IN CURVED TUBE TO THAT IN STRAIGHT TUBE.

      QR=0.
      DO 49 J=1,NA
49  QR=QR+W(1,J,3)+W(2,J,3)+W(2,J+1,3)
      QR=QR*CO(1)
      DO 249 I=2,NR
      S=0.
      DO 149 J=1,NA
149 S=S+W(I,J,3)+W(I+1,J,3)+W(I,J+1,3)+W(I+1,J+1,3)
249 QR=QR+CO(I)*S
C     PRINT OUT VALUE OF FLUX RATIO JUST COMPUTED.

      WRITE(6,90) QR

C     WRITE SOLUTION TO FILE IF INPUT CONTROL
C     PARAMETER IPCH SO DICTATES, AND PRINT MESSAGE INDICATING THAT
C     THIS FILE WRITING WAS DONE.

C     NTB 2025:
C     Write solution to file
      IF (IPCH.NE.0) THEN
        WRITE(file_id, '(i0)') INT(D)
        file_name = 'file_D' // TRIM(ADJUSTL(file_id)) // '.dat'
        PRINT*,"file name is ",file_name
        OPEN(1,FILE = file_name,STATUS = 'replace')
        WRITE(1,*) NRP1
        WRITE(1,*) NAP1
        WRITE(1,*) XI
        WRITE(1,*) RHO
        WRITE(1,*) EPS
        WRITE(1,*) D
        WRITE(1,*) QR
        DO 350 I=1,NRP1
350  WRITE(1,*) PHI(I,:,3)
```

```
      DO 360 I=1,NRP1
360    WRITE(1,*) W(I,:,3)
      DO 370 I=1,NRP1
370    WRITE(1,*) OMEGA(I,:,3)
      CLOSE(1)
      END IF

      IF(IPCH.NE.0) WRITE(6,86)

C     TEST WHETHER OR NOT SOLUTION JUST OBTAINED IS TO BE SAVED IN
C     MEMORY FOR SOME SUCCEEDING CASE WITH A DIFFERENT VALUE OF D.

      IF(ISAVE.EQ.0) GO TO 5

C     SAVE CURRENT SOLUTION IN ANOTHER AREA OF MEMORY.

      DO 50 I=1,NRP1
      DO 50 J=1,NAP1
      PHI(I,J,4)=PHI(I,J,3)
      W(I,J,4)=W(I,J,3)
50    OMEGA(I,J,4)=OMEGA(I,J,3)

C     SAVE VALUE OF D FOR WHICH SOLUTION WAS JUST OBTAINED IN DSTART.

      DSTART=D
      GO TO 5

C     CONVERGENCE FAILURE MESSAGES

C     PHI ITERATION FAILED. READ NEXT DATA CASE.

55    WRITE(6,96)
      GO TO 5

C     W ITERATION FAILED. REDUCE OVER-RELAXATION FACTOR AND TRY AGAIN,
C     UNLESS THIS HAS ALREADY BEEN DONE THE MAXIMUM NUMBER OF TIMES
C     (NOR), IN WHICH CASE READ NEXT DATA CASE.

56    WRITE(6,95) RHO(2)
      IF(IRW.GE.NOR) GO TO 5
      IRW=IRW+1
      RHO(2)=RHO(2)-DOR(IRW)
      RHOC(2)=1.-RHO(2)
      DO 156 I=1,NRP1
      DO 156 J=1,NAP1
156    W(I,J,3)=W(I,J,1)
      GO TO 26

C     OMEGA ITERATION FAILED. REDUCE O-R FACTOR AND TRY AGAIN, UNLESS
C     THIS HAS ALREADY BEEN DONE THE MAXIMUM NUMBER OF TIMES (NOR),
C     IN WHICH CASE READ THE NEXT DATA CASE.

57    WRITE(6,94) RHO(3)
```

```
      IF(IRO.GE.NOR) GO TO 5
      IRO=IRO+1
      RHO(3)=RHO(3)-DOR(IRO)
      RHOC(3)=1.-RHO(3)
      DO 157 I=1,NRP1
      DO 157 J=1,NAP1
157  OMEGA(I,J,3)=OMEGA(I,J,1)
      GO TO 41

C      OUTER ITERATION FAILED. PRINT MESSAGE INDICATING THIS AND READ
C      NEXT DATA CASE.

      58 WRITE(6,93)
      GO TO 5

C      TERMINATION POINT FOR PROGRAM. CONTROL REACHES HERE AFTER ATTEMPT
C      TO READ PAST LAST DATA RECORD.

      70 STOP

      98 FORMAT(6X 'D =' F5.0//13X 'PHI' 7X 'W' 5X 'OMEGA' // 5X
      * 'RHO =' F6.2,2(3X F6.2)/6X 'XI =' F6.4,3(3X F6.4)/5X 'EPS ='
      * F6.4,2(3X F6.4))
      97 FORMAT('OUTER ITERATION CONVERGED TO GIVEN TOLERANCES.')
      96 FORMAT('SOR FOR PHI FAILED.')
      95 FORMAT('SOR FOR W FAILED WITH SOR FACTOR =' F6.2)
      94 FORMAT('SOR FOR OMEGA FAILED WITH SOR FACTOR =' F6.2)
      93 FORMAT('OUTER ITERATION FAILED TO CONVERGE.')
      92 FORMAT('//OUTER ITERATION' I5//)
      90 FORMAT('FLUX RATIO =' F10.5)
      87 FORMAT('INITIAL ITERATE TAKEN FROM SOLUTION FOR D =' F7.0)
      86 FORMAT('SOLUTION WAS OUTPUT TO FILE.')

      END

C      THIS ROUTINE PRINTS OUT A DISCRETE FUNCTION IN A RECTANGULAR
C      FORMAT WHICH IS RELATED IN THE FOLLOWING WAY TO THE POLAR GRID
C      IMPOSED ON THE PHYSICAL REGION.
C      THE TOP LINE OF THE PRINT BLOCK CORRESPONDS TO VALUES OF THE
C      FUNCTION ALONG THE ARC R = 1.
C      THE BOTTOM LINE OF THE PRINT BLOCK CORRESPONDS TO THE VALUE
C      OF THE FUNCTION AT THE ORIGIN (R = 0).

C      NTB 2025:
C      Added x (NR) and y (NA) as inputs
      SUBROUTINE OUTPUT(VAR,ISOR,A,x,y)

C      NTB 2025:
C      Added definitions
      CHARACTER*5 APHI, AW, AOMEGA, VAR
      INTEGER :: x,y
      DIMENSION A(x+1,y+1)
      NRP1=x+1
```

```
NAP1=y+1

DATA APHI /'PHI'/, AW /'W      '/, AOMEGA /'OMEGA'/
99 FORMAT(A6,5X I5,2X 'SOR ITERATIONS'/)
98 FORMAT(1X F6.2,17F7.2,F6.2)
97 FORMAT(1X F6.1,17F7.1,F6.1)
KRP1=NRP1
KAP1=NAP1
WRITE(6,99) VAR,ISOR
IF(VAR.NE.APHI) GO TO 8
DO 5 I=1,NRP1
5 WRITE(6,98) (A(KRP1-I+1,KAP1-J+1),J=1,KAP1)
RETURN
8 DO 10 I=1,NRP1
10 WRITE(6,97) (A(KRP1-I+1,KAP1-J+1),J=1,KAP1)
RETURN
END
```

Appendix C. Complete modern Fortran code

This appendix includes the completely modernised Fortran version of the original FORTRAN 66 code written by A.B.Schubert in 1972 [4].

Compiling and running the solution creates a separate file for each value of D .

```
! Appendix: FORTRAN Program for Secondary Flow, by A. B. Schubert (1972)
```

```
!
```

```
! Nils T. Basse (2025):
```

```
! Complete modern Fortran version created with the assistance of GitHub ↗  
! Copilot (ChatGPT-4.1).
```

```
MODULE KIND_MOD
```

```
!-----  
!> Defines the kind parameter for double precision real numbers (dp).  
!! This ensures consistent numerical precision throughout the program.  
!! dp = SELECTED_REAL_KIND(15, 307) corresponds to at least 15 decimal  
!! digits of precision and an exponent range of at least 10±307.  
!-----
```

```
IMPLICIT NONE
```

```
INTEGER, PARAMETER :: dp = SELECTED_REAL_KIND(15, 307)
```

```
END MODULE KIND_MOD
```

```
MODULE ERROR_MOD
```

```
USE KIND_MOD
```

```
IMPLICIT NONE
```

```
CONTAINS
```

```
!-----  
!> Handles error codes and prints appropriate error messages.  
!! code : Error code (integer)  
!! val  : Optional value, used for some error messages (real)  
!-----
```

```
SUBROUTINE ERROR_HANDLER(code, val)
```

```
INTEGER, INTENT(IN) :: code
```

```
REAL(KIND=dp), OPTIONAL, INTENT(IN) :: val
```

```
SELECT CASE (code)
```

```
  CASE (55)
```

```
    WRITE(*, '("SOR FOR PHI FAILED.")')
```

```
  CASE (56)
```

```
    IF (PRESENT(val)) THEN
```

```
      WRITE(*, '("SOR FOR W FAILED WITH SOR FACTOR =", F6.2)') val
```

```
    ELSE
```

```
      WRITE(*, '("SOR FOR W FAILED WITH SOR FACTOR =", F6.2)') 0.0_dp
```

```
    END IF
```

```
  CASE (57)
```

```
    IF (PRESENT(val)) THEN
```

```
      WRITE(*, '("SOR FOR OMEGA FAILED WITH SOR FACTOR =", F6.2)') val
```

```
    ELSE
```

```
      WRITE(*, '("SOR FOR OMEGA FAILED WITH SOR FACTOR =", F6.2)') 0.0_dp
```

```
    END IF
```

```
  CASE (58)
```

```
    WRITE(*, '("OUTER ITERATION FAILED TO CONVERGE.")')
```

```
  CASE DEFAULT
```

```
    WRITE(*,*) 'Unknown error code in ERROR_HANDLER.'
```

```
END SELECT
```

```
END SUBROUTINE ERROR_HANDLER
```

```
END MODULE ERROR_MOD
```

```

MODULE OUTPUT_MOD
  USE KIND_MOD
  IMPLICIT NONE
CONTAINS
  !-----
  !> Outputs the solution array for a given variable (PHI, W, or OMEGA).
  !! VAR   : Name of variable to output ('PHI ', 'W   ', 'OMEGA')
  !! ISOR  : Number of SOR iterations performed
  !! A     : Solution array to output
  !! x, y  : Grid dimensions
  !-----
  SUBROUTINE OUTPUT(VAR, ISOR, A, x, y)
    CHARACTER(LEN=5), INTENT(IN) :: VAR
    INTEGER, INTENT(IN) :: x, y, ISOR
    REAL(KIND=dp), INTENT(IN) :: A(x+1, y+1)

    CHARACTER(LEN=5), PARAMETER :: APhi='PHI ', AW='W   ', AOmega='OMEGA'
    INTEGER :: I, J, NRP1, NAP1

    NRP1 = x + 1
    NAP1 = y + 1

    WRITE(*, '(A6,5X,I5,2X,"SOR ITERATIONS"/)') VAR, ISOR

    ! Output array in reverse order for both indices for better
    ! visualization
    IF (VAR /= APhi) THEN
      DO I = 1, NRP1
        WRITE(*, '(1X,F6.1,17F7.1,F6.1)') (A(NRP1 - I + 1, NAP1 - J + 1),
          J = 1, NAP1)
      END DO
    ELSE
      DO I = 1, NRP1
        WRITE(*, '(1X,F6.2,17F7.2,F6.2)') (A(NRP1 - I + 1, NAP1 - J + 1),
          J = 1, NAP1)
      END DO
    END IF

  END SUBROUTINE OUTPUT
END MODULE OUTPUT_MOD

MODULE SOR_MOD
  USE KIND_MOD
  IMPLICIT NONE
CONTAINS
  !-----
  !> Performs SOR iteration for PHI variable.
  ! Updates PHI using SOR until convergence or maximum iterations.
  !-----
  SUBROUTINE SOR_PHI(PHI, OMEGA, B, C, RHOC, RHO, EPPS, ISOR_PHI, MAXSOR,
    NR, NA, NRP1, NAP1, NRM1, ERROR_HANDLER)
    IMPLICIT NONE
    INTEGER, INTENT(IN) :: NR, NA, NRP1, NAP1, NRM1, MAXSOR

```

```

REAL(KIND=dp), INTENT(INOUT) :: PHI(NRP1, NAP1, 4), OMEGA(NRP1, NAP1, 4)
REAL(KIND=dp), INTENT(IN) :: B(NRP1, 5), RHOC(3), RHO(3), EPPS(3)
REAL(KIND=dp), INTENT(INOUT) :: C(NRP1, NAP1)
INTEGER, INTENT(OUT) :: ISOR_PHI
INTERFACE
  SUBROUTINE ERROR_HANDLER(code, val)
    USE KIND_MOD
    INTEGER, INTENT(IN) :: code
    REAL(KIND=dp), OPTIONAL, INTENT(IN) :: val
  END SUBROUTINE ERROR_HANDLER
END INTERFACE
INTEGER :: I, J, ICONV

ISOR_PHI = 0
PHI_SOR: DO
  ICONV = 0
  ! ICONV is a local SOR convergence flag: set to 1 if any update
  ! exceeds tolerance.
  ! Update PHI at interior points (2 <= r <= NR-1, 2 <= alpha <= NA)
  DO I = 2, NRM1
    DO J = 2, NA
      PHI(I,J,3) = RHOC(1)*PHI(I,J,2) + RHO(1)*( &
        B(I,1)*PHI(I+1,J,2) + B(I,2)*PHI(I,J+1,2) &
        + B(I,3)*PHI(I-1,J,3) + B(I,4)*PHI(I,J-1,3) + C(I,J))
      IF (ABS(PHI(I,J,2) - PHI(I,J,3)) > EPPS(1)) ICONV = 1
    END DO
  END DO
  ! Update PHI at the outer boundary (r = NR)
  DO J = 2, NA
    PHI(NR,J,3) = RHOC(1)*PHI(NR,J,2) + RHO(1)*0.25_dp*PHI(NRM1,J,3)
    IF (ABS(PHI(NR,J,2) - PHI(NR,J,3)) > EPPS(1)) ICONV = 1
  END DO
  ! Check for convergence: if no updates exceed tolerance, exit SOR loop
  IF (ICONV == 0) EXIT PHI_SOR
  ISOR_PHI = ISOR_PHI + 1
  IF (ISOR_PHI >= MAXSOR) THEN
    ! PHI iteration failed: skip to next case
    CALL ERROR_HANDLER(55)
    EXIT PHI_SOR
  END IF
  ! Prepare for next SOR iteration by copying new values to previous
  ! iterate
  PHI(2:NR,2:NA,2) = PHI(2:NR,2:NA,3)
END DO PHI_SOR
END SUBROUTINE SOR_PHI

!-----
!> Performs SOR iteration for W variable, with possible reduction of
! over-relaxation factor if convergence fails.
!-----
SUBROUTINE SOR_W(W, E, EPPS2, RHOC2, RHO2, ISOR_W, MAXSOR, NR, NA, NRP1, >
  NAP1, NAH, E1, E2, E3, E4, DOR, NOR, IRW, ICONV, ERROR_HANDLER)
  IMPLICIT NONE

```

```
INTEGER, INTENT(IN) :: NR, NA, NRP1, NAP1, NAH, MAXSOR, NOR
REAL(KIND=dp), INTENT(INOUT) :: W(NRP1, NAP1, 4)
REAL(KIND=dp), INTENT(IN) :: E(NRP1, NAP1, 6)
REAL(KIND=dp), INTENT(IN) :: EPPS2, RHOC2
REAL(KIND=dp), INTENT(INOUT) :: RH02
REAL(KIND=dp), INTENT(IN) :: E1, E2, E3, E4
REAL(KIND=dp), INTENT(IN) :: DOR(NOR)
INTEGER, INTENT(INOUT) :: IRW
INTEGER, INTENT(OUT) :: ISOR_W, ICONV
INTERFACE
  SUBROUTINE ERROR_HANDLER(code, val)
    USE KIND_MOD
    INTEGER, INTENT(IN) :: code
    REAL(KIND=dp), OPTIONAL, INTENT(IN) :: val
  END SUBROUTINE ERROR_HANDLER
END INTERFACE

INTEGER :: I, J
REAL(KIND=dp) :: RHOC2_LOCAL, RH02_LOCAL

ISOR_W = 0
RHOC2_LOCAL = RHOC2
RH02_LOCAL = RH02

W_SOR: DO
  ! If maximum SOR iterations reached, reduce RH02 and retry if allowed
  IF (ISOR_W .GE. MAXSOR) THEN
    ! W iteration failed: reduce over-relaxation factor and try again
    CALL ERROR_HANDLER(56, RH02_LOCAL)
    IF (IRW .GE. NOR) EXIT W_SOR
    IRW = IRW + 1
    RH02_LOCAL = RH02_LOCAL - DOR(IRW)
    RHOC2_LOCAL = 1.0_dp - RH02_LOCAL
    DO I = 1, NRP1
      DO J = 1, NAP1
        W(I, J, 3) = W(I, J, 1)
      END DO
    END DO
    ISOR_W = 0
    CYCLE W_SOR
  END IF

  ISOR_W = ISOR_W + 1

  ! Update previous SOR iterates for W
  W(1, 1, 2) = W(1, 1, 3)
  DO I = 2, NR
    DO J = 1, NAP1
      W(I, J, 2) = W(I, J, 3)
    END DO
  END DO

  ICONV = 0
```

```

! ICONV is a local SOR convergence flag: set to 1 if any update
! exceeds tolerance.
! Update W at the origin (r=0)
W(1, 1, 3) = RHOC2_LOCAL * W(1, 1, 2) + RHO2_LOCAL * (E1 * W(2, 1, 2)
+ E2 * W(2, NAH, 2) + E3 * W(2, NAP1, 2) + E4)
DO J = 2, NAP1
  W(1, J, 3) = W(1, 1, 3)
END DO
IF (ABS(W(1, 1, 2) - W(1, 1, 3)) .GT. EPPS2) ICONV = 1

! Update W along alpha=0 (excluding r=0 and r=1)
DO I = 2, NR
  W(I, 1, 3) = RHOC2_LOCAL * W(I, 1, 2) + RHO2_LOCAL * (E(I, 1, 1) * W
(I + 1, 1, 2) + E(I, 1, 2) * 2.0_dp * W(I, 2, 2) + E(I, 1, 3) * W
(I - 1, 1, 3) + E(I, 1, 5))
  IF (ABS(W(I, 1, 3) - W(I, 1, 2)) .GT. EPPS2) ICONV = 1
END DO

! Update W in the interior (2 <= r <= NR, 2 <= alpha <= NA)
DO I = 2, NR
  DO J = 2, NA
    W(I, J, 3) = RHOC2_LOCAL * W(I, J, 2) + RHO2_LOCAL * (E(I, J,
1) * W(I + 1, J, 2) + E(I, J, 2) * W(I, J + 1, 2) + E(I, J,
3) * W(I - 1, J, 3) + E(I, J, 4) * W(I, J - 1, 3) + E(I, J, 5))
    IF (ABS(W(I, J, 2) - W(I, J, 3)) .GT. EPPS2) ICONV = 1
  END DO
END DO

! Update W along alpha=pi (excluding r=0 and r=1)
DO I = 2, NR
  W(I, NAP1, 3) = RHOC2_LOCAL * W(I, NAP1, 2) + RHO2_LOCAL * (E(I,
NAP1, 1) * W(I + 1, NAP1, 2) + E(I, NAP1, 3) * W(I - 1, NAP1,
3) + 2.0_dp * E(I, NAP1, 4) * W(I, NA, 3) + E(I, NAP1, 5))
  IF (ABS(W(I, NAP1, 2) - W(I, NAP1, 3)) .GT. EPPS2) ICONV = 1
END DO

! Check for convergence: if no updates exceed tolerance, exit SOR loop
IF (ICONV .EQ. 0) EXIT W_SOR
END DO W_SOR

! Update RHO2 in caller if changed
RHO2 = RHO2_LOCAL

END SUBROUTINE SOR_W

!-----
!> Performs SOR iteration for OMEGA variable.
! Updates OMEGA using SOR until convergence or maximum iterations.
!-----
SUBROUTINE SOR_OMEGA(OMEGA, E, RHOC3, RHO3, EPPS3, ISOR_OMEGA, MAXSOR,
NR, NA, NRP1, NAP1, ICONV, ERROR_HANDLER)
IMPLICIT NONE

```

```

INTEGER, INTENT(IN) :: NR, NA, NRP1, NAP1, MAXSOR
REAL(KIND=dp), INTENT(INOUT) :: OMEGA(NRP1, NAP1, 4)
REAL(KIND=dp), INTENT(IN) :: E(NRP1, NAP1, 6)
REAL(KIND=dp), INTENT(IN) :: RHOC3, RHO3, EPPS3
INTEGER, INTENT(INOUT) :: ISOR_OMEGA
INTEGER, INTENT(OUT) :: ICONV
INTERFACE
  SUBROUTINE ERROR_HANDLER(code, val)
    USE KIND_MOD
    INTEGER, INTENT(IN) :: code
    REAL(KIND=dp), OPTIONAL, INTENT(IN) :: val
  END SUBROUTINE ERROR_HANDLER
END INTERFACE
INTEGER :: I, J

ISOR_OMEGA = 0
OMEGA_SOR: DO
  ! If maximum SOR iterations reached, call error handler and exit
  IF (ISOR_OMEGA .GE. MAXSOR) THEN
    CALL ERROR_HANDLER(57, RHO3)
    EXIT OMEGA_SOR
  END IF

  ISOR_OMEGA = ISOR_OMEGA + 1

  ! Update previous SOR iterates for OMEGA
  OMEGA(2:NR, 2:NA, 2) = OMEGA(2:NR, 2:NA, 3)

  ICONV = 0

  ! ICONV is a local SOR convergence flag: set to 1 if any update
  ! exceeds tolerance.
  ! Update OMEGA at interior points (2 <= r <= NR, 2 <= alpha <= NA)
  DO I = 2, NR
    DO J = 2, NA
      OMEGA(I, J, 3) = RHOC3 * OMEGA(I, J, 2) + RHO3 * ( &
        E(I, J, 1) * OMEGA(I + 1, J, 2) &
        + E(I, J, 2) * OMEGA(I, J + 1, 2) &
        + E(I, J, 3) * OMEGA(I - 1, J, 3) &
        + E(I, J, 4) * OMEGA(I, J - 1, 3) &
        + E(I, J, 6))
      IF (ABS(OMEGA(I, J, 2) - OMEGA(I, J, 3)) .GT. EPPS3) ICONV = 1
    END DO
  END DO

  ! Check for convergence: if no updates exceed tolerance, exit SOR loop
  IF (ICONV .EQ. 0) EXIT OMEGA_SOR
END DO OMEGA_SOR
END SUBROUTINE SOR_OMEGA

!-----
!> Smooths the solution array for a given variable.
! Used to blend new and previous iterates and check for convergence.

```

```

C:\Fortran codes\Schubert_1972_complete_modern_Fortran.f90 7
! Smoothing blends new and previous iterates to damp oscillations and
! improve convergence.
!-----
SUBROUTINE SMOOTH(N1, N2, ARR, XI, XIC, EPS, ICV)
  INTEGER, INTENT(IN) :: N1, N2
  REAL(KIND=dp), INTENT(INOUT) :: ARR(N1, N2, 4)
  REAL(KIND=dp), INTENT(IN) :: XI, XIC, EPS
  INTEGER, INTENT(INOUT) :: ICV
  INTEGER :: I, J
  ! Smooth the solution to stabilize convergence and update the
  ! convergence flag.
  DO I = 2, N1-1
    DO J = 2, N2-1
      ARR(I,J,3) = XI*ARR(I,J,1) + XIC*ARR(I,J,3)
      IF (ABS(ARR(I,J,1)-ARR(I,J,3)) > EPS) ICV = 1
    END DO
  END DO
END SUBROUTINE SMOOTH
END MODULE SOR_MOD

!-----
!> Main program: Solves for flow in a curved tube using SOR and outer
!! iteration methods. Handles multiple cases for different D values.
!-----
PROGRAM MAIN
  USE KIND_MOD
  USE ERROR_MOD
  USE OUTPUT_MOD
  USE SOR_MOD
  IMPLICIT NONE

!----- Parameter and variable declarations -----
! Define grid, solution, iteration parameters and arrays

! PHI(:, :, 1): previous outer iterate (from last outer iteration)
! PHI(:, :, 2): previous SOR iterate (from last SOR sweep)
! PHI(:, :, 3): current SOR iterate (being updated in this sweep)
! PHI(:, :, 4): storage for next case (used as initial guess for next D)
! The same convention applies to W and OMEGA arrays.

! B(I,1): east (I+1), B(I,2): north (J+1), B(I,3): west (I-1), B(I,4):
! south (J-1), B(I,5): source term
! E(I,J,1): east (I+1), E(I,J,2): north (J+1), E(I,J,3): west (I-1), E
! (I,J,4): south (J-1), E(I,J,5): source, E(I,J,6): denominator

INTEGER, PARAMETER :: NR = 2*10, NA = 2*18 ! Number of radial and angular
grid points
INTEGER, PARAMETER :: NRP1 = NR + 1, NAP1 = NA + 1 ! Number of grid
points including boundaries
INTEGER, PARAMETER :: NRM1 = NR - 1 ! Number of radial grid points
excluding boundaries
INTEGER, PARAMETER :: NAH = NA / 2 + 1 ! Halfway point in angular grid
INTEGER, PARAMETER :: NPHI = 4 ! Number of storage slots for PHI, W,

```

```

    OMEGA (e.g., for time levels or SOR iterates)
    INTEGER, PARAMETER :: NB = 5      ! Number of B coefficients for SOR of PHI
    INTEGER, PARAMETER :: NE = 6      ! Number of E coefficients for SOR of W/ OMEGA
    OMEGA
    INTEGER :: clock_start, clock_end, clock_rate ! For timing the execution
    REAL(KIND=dp) :: elapsed_time ! Elapsed CPU time for the program execution

    REAL(KIND=dp) :: PHI(NRP1, NAP1, NPHI) ! Stream function solution
    array, indexed by (radial, angular, SOR iterate/time-level)
    REAL(KIND=dp) :: W(NRP1, NAP1, NPHI) ! Axial velocity solution array,
    indexed by (radial, angular, SOR iterate/time-level)
    REAL(KIND=dp) :: OMEGA(NRP1, NAP1, NPHI) ! Vorticity solution array,
    indexed by (radial, angular, SOR iterate/time-level)
    REAL(KIND=dp) :: SA(NAP1) ! Sine of angular grid points (used in
    advection terms and geometry)
    REAL(KIND=dp) :: COSA(NAP1) ! Cosine of angular grid points (used in
    advection terms and geometry)
    REAL(KIND=dp) :: RINV(NRP1) ! 1/r for each radial grid point (used in
    cylindrical coordinate terms)
    REAL(KIND=dp) :: RINV2(NRP1) ! (1/r)^2 for each radial grid point (used
    in Laplacian and advection terms)
    REAL(KIND=dp) :: CA(NRP1, NAP1) ! Coefficient array for SOR, used in
    OMEGA update to account for geometry (cosine-weighted term in advection)
    REAL(KIND=dp) :: DELA(NR) ! Area of each radial shell, used in
    flux calculation
    REAL(KIND=dp) :: CO(NR) ! Scaling factor for each shell in the
    flux calculation
    REAL(KIND=dp) :: B(NRP1, NB) ! SOR coefficients for PHI update: (1)
    east, (2) north, (3) west, (4) south, (5) source
    REAL(KIND=dp) :: C(NRP1, NAP1) ! Source term (right-hand side) for
    PHI SOR update, computed from OMEGA
    REAL(KIND=dp) :: E(NRP1, NAP1, NE) ! SOR coefficients for W/OMEGA: (1)
    east, (2) north, (3) west, (4) south, (5) source, (6) denominator
    REAL(KIND=dp) :: EE(NRP1) ! Intermediate denominator for SOR
    coefficients for W
    REAL(KIND=dp) :: EF(NRP1) ! Used in boundary SOR coefficient
    calculations for W (north/south stencils)
    REAL(KIND=dp) :: EE1, EE2 ! EE1: Denominator for SOR
    coefficients at lower angular boundary (alpha=0); EE2: at upper angular
    boundary (alpha=pi)
    REAL(KIND=dp) :: XI(4) ! Smoothing factors: (1) PHI, (2) W,
    (3) OMEGA, (4) unused/extra
    REAL(KIND=dp) :: XIC(4) ! Complement of smoothing factors: XIC
    (i) = 1 - XI(i)
    REAL(KIND=dp) :: RHO(3) ! SOR over-relaxation factors: (1)
    PHI, (2) W, (3) OMEGA
    REAL(KIND=dp) :: RHOC(3) ! Complement of SOR factors: RHOC(i) =
    1 - RHO(i)
    REAL(KIND=dp) :: EPS(3) ! Convergence tolerances: (1) PHI, (2)
    W, (3) OMEGA
    REAL(KIND=dp) :: EPPS(3) ! Tighter SOR convergence tolerances:
    EPPS(i) = 0.05 * EPS(i)
    REAL(KIND=dp) :: BO ! Denominator for SOR coefficient calculation

```

```
    for PHI
REAL(KIND=dp) :: CON      ! Scaling constant for flux calculation
REAL(KIND=dp) :: DDR2    ! D * DR^2, used in SOR coefficient
calculations
REAL(KIND=dp) :: DDRDAM  ! D * DR * DA, used in SOR coefficient
calculations
REAL(KIND=dp) :: DRRH    ! Half radial grid spacing times inverse
radius, used in SOR coefficient calculations
REAL(KIND=dp) :: DR      ! Radial grid spacing
REAL(KIND=dp) :: DA      ! Angular grid spacing
REAL(KIND=dp) :: DAH     ! Half angular grid spacing
REAL(KIND=dp) :: DRH     ! Half radial grid spacing
REAL(KIND=dp) :: DRDAM   ! Product of DR and DA
REAL(KIND=dp) :: DRDA    ! Ratio of DR to DA
REAL(KIND=dp) :: DADR    ! Ratio of DA to DR
REAL(KIND=dp) :: E0      ! Denominator for SOR coefficients for W at
the origin
REAL(KIND=dp) :: E1, E2, E3, E4 ! SOR coefficients for W at the origin
and boundaries
REAL(KIND=dp) :: GAMMA   ! Central difference in PHI in the radial
direction (used for upwinding)
REAL(KIND=dp) :: DELTA   ! Central difference in PHI in the angular
direction (used for upwinding)
REAL(KIND=dp) :: DELTA1  ! PHI difference at lower angular boundary
REAL(KIND=dp) :: DELTA2  ! PHI difference at upper angular boundary
REAL(KIND=dp) :: QR      ! Total flux ratio for the current case, computed
by integrating the W velocity field over the domain
REAL(KIND=dp) :: S       ! Local sum of W values for a grid cell, used as
part of the flux calculation for each radial shell
REAL(KIND=dp) :: PI      ! Value of pi, used for grid spacing and geometry
calculations
REAL(KIND=dp) :: D       ! Dean number for the current case
REAL(KIND=dp) :: DSTART  ! Dean number from the previous case (used for
reporting initial guess source)
REAL(KIND=dp) :: DOR(3)  ! Decrement values for SOR over-relaxation
factor when convergence fails
INTEGER :: I, J          ! Loop counters for radial (I) and angular (J) indices
INTEGER :: ICONV        ! SOR convergence flag: set to 1 if any SOR update
exceeds tolerance in the current SOR loop (PHI, W, or OMEGA)
INTEGER :: ICV          ! Outer iteration convergence flag: set to 1 if any
variable has not converged.
INTEGER :: IOUT         ! Outer iteration counter: counts the number of outer
(nonlinear) iterations for the current case
INTEGER :: IRW          ! W SOR relaxation reduction counter: counts the
number of times the SOR over-relaxation factor for W has been reduced
due to lack of convergence
INTEGER :: IRO          ! OMEGA SOR relaxation reduction counter: counts the
number of times the SOR over-relaxation factor for OMEGA has been
reduced due to lack of convergence
INTEGER :: ctr          ! Main loop index for iterating over Dean number cases
(1 to 7)
INTEGER :: MAXSOR       ! Maximum number of SOR (Successive Over-Relaxation)
iterations allowed for PHI, W, or OMEGA before triggering error
```

```

handling or relaxation reduction
INTEGER :: MAXOUT ! Maximum number of outer (nonlinear) iterations
allowed for each Dean number case before giving up on convergence
INTEGER :: NOR ! Maximum number of times the SOR over-relaxation
factor can be reduced for W or OMEGA before giving up on convergence
INTEGER :: ISAVE ! If nonzero, save the current solution at the end of
each case to use as the initial guess for the next case.
INTEGER :: IUSE ! If nonzero, use the saved solution from the previous
case as the initial guess for the current case.
INTEGER :: IFIL ! Output-to-file flag; if nonzero, writes solution and
parameters to a file for each case
INTEGER :: ISOR_PHI ! Number of SOR iterations performed for PHI in the
current outer iteration (reset each time PHI is solved)
INTEGER :: ISOR_W ! Number of SOR iterations performed for W in the
current outer iteration (reset each time W is solved)
INTEGER :: ISOR_OMEGA ! Number of SOR iterations performed for OMEGA in
the current outer iteration (reset each time OMEGA is solved)
CHARACTER(LEN=10) :: file_id ! String version of Dean number D for file
naming
CHARACTER(LEN=50) :: file_name ! Output file name for current case,
constructed from file_id
INTEGER :: unit
LOGICAL :: FAILED ! Set to .TRUE. if the outer (nonlinear) iteration
fails to converge (IOUT >= MAXOUT)

CALL SYSTEM_CLOCK(COUNT_RATE=clock_rate)
CALL SYSTEM_CLOCK(COUNT=clock_start)

!----- Initialization and setup -----
! Initialize all solution arrays (PHI, W, OMEGA) to zero to ensure a clean
start and avoid uninitialized values.
PHI(:, :, :) = 0.0_dp
W(:, :, :) = 0.0_dp
OMEGA(:, :, :) = 0.0_dp

PI = 3.14159255_dp
MAXSOR = 2500
MAXOUT = 600
NOR = 3
DOR = (/0.2_dp, 0.2_dp, 0.2_dp/)

DR = 1._dp / NR
DA = PI / NA
DAH = .5_dp * DA
DRH = .5_dp * DR
DRDAM = DR * DA
DRDA = DR / DA
DADR = DA / DR

! Compute area elements for flux calculation (used in QR calculation)
DO I = 1, NR
  DELA(I) = (2 * I - 1) * DRH * DRDAM
END DO

```

```
! Compute trigonometric and radius-dependent arrays for the grid
```

```
SA(1) = 0._dp
DO J = 2, NA
  SA(J) = SIN((J - 1) * DA) * DAH
  COSA(J) = COS((J - 1) * DA)
END DO
SA(NAP1) = 0._dp
COSA(NAP1) = -1._dp
RINV(NRP1) = 1._dp
RINV2(NRP1) = 1._dp
DO I = 2, NR
  RINV(I) = 1._dp / ((I - 1) * DR)
  DRRH = DRH * RINV(I)
  RINV2(I) = RINV(I) ** 2
  DO J = 2, NA
    CA(I, J) = DRRH * COSA(J)
  END DO
END DO
```

```
!----- SOR coefficient setup for PHI -----
```

```
! Set up finite-difference stencil coefficients for SOR update of PHI.
! Each coefficient corresponds to a neighbor direction or the source term.
```

```
DO I = 2, NR
  BO = 2._dp * (DADR + DRDA * RINV2(I)) + DA * RINV(I)
  B(I, 1) = (DADR + DA * RINV(I)) / BO
  B(I, 2) = DRDA * RINV2(I) / BO
  B(I, 3) = DADR / BO
  B(I, 4) = B(I, 2)
  B(I, 5) = DRDAM / BO
END DO
```

```
!----- Main case loop over D values -----
```

```
! Main loop over Dean number cases.
! For each case, set parameters, solve the system, and output results.
ISAVE = 1
IUSE = 1
IFIL = 1
```

```
DO ctr = 1, 7
```

```
! Set D, EPS, RHO, XI for this case (D: Dean number, EPS: tolerances,
RHO: SOR factors, XI: smoothing factors) →
```

```
SELECT CASE (ctr)
```

```
  CASE (1)
```

```
    D = 10_dp
    EPS = (/1.0E-5_dp, 1.0E-3_dp, 1.0E-4_dp/)
    RHO = (/1.5_dp, 1.8_dp, 1.5_dp/)
    XI = 0.1_dp
```

```
  CASE (2)
```

```
    D = 100_dp
    EPS = (/2.0E-4_dp, 5.0E-3_dp, 5.0E-3_dp/)
    RHO = (/1.5_dp, 1.7_dp, 1.5_dp/)
    XI = 0.1_dp
```

```

CASE (3)
  D = 250_dp
  EPS = (/2.0E-3_dp, 2.0E-2_dp, 4.0E-2_dp/)
  RHO = (/1.5_dp, 1.5_dp, 1.5_dp/)
  XI = 0.1_dp
CASE (4)
  D = 500_dp
  EPS = (/4.0E-3_dp, 4.0E-2_dp, 8.0E-2_dp/)
  RHO = (/1.5_dp, 1.5_dp, 1.5_dp/)
  XI = 0.1_dp
CASE (5)
  D = 1000_dp
  EPS = (/5.0E-3_dp, 5.0E-2_dp, 17.0E-2_dp/)
  RHO = (/1.5_dp, 1.5_dp, 1.5_dp/)
  XI = (/0.5_dp, 0.1_dp, 0.1_dp, 0.5_dp/)
CASE (6)
  D = 2000_dp
  EPS = (/7.0E-3_dp, 8.0E-2_dp, 3.0E-1_dp/)
  RHO = (/0.8*1.5_dp, 0.8*1.5_dp, 0.8*1.3_dp/)
  XI = (/0.5_dp, 0.1_dp, 0.1_dp, 0.5_dp/)
CASE (7)
  D = 5000_dp
  EPS = (/1.0E-2_dp, 15.0E-2_dp, 6.0E-1_dp/)
  RHO = (/0.8*1.5_dp, 0.8*1.5_dp, 0.8*1.3_dp/)
  XI = (/0.3_dp, 0.1_dp, 0.1_dp, 0.7_dp/)
END SELECT

! Print out input parameters for this case
WRITE(*, '(6X,"D =",F5.0,//13X,"PHI",7X,"W",5X,"OMEGA"//5X,"RHO =",F6.2,2 >
(3X,F6.2)/6X,"XI =",F6.4,3(3X,F6.4)/5X,"EPS =",F6.4,2(3X,F6.4))' ) D, >
RHO, XI, EPS

! Compute D-dependent parameters for flux and SOR
DDRDM = D * DRDM
DDR2 = D * DR ** 2
CON = 4._dp / (PI * D)
CO(1) = 16._dp * DELA(1) / (3._dp * PI * D)
DO I = 2, NR
  CO(I) = CON * DELA(I)
END DO

! Compute complements and tolerances for smoothing and SOR
DO I = 1, 3
  XIC(I) = 1._dp - XI(I)
  RHOC(I) = 1._dp - RHO(I)
  EPPS(I) = .05_dp * EPS(I)
END DO
XIC(4) = 1._dp - XI(4)

!----- Initial guess for outer iterates -----
! If IUSE /= 0, use previous solution as initial guess for current case.
! Otherwise, initialize all outer iterates to zero for a fresh start.

```

```

IF (IUSE .NE. 0) THEN
  PHI(:, :, 3) = PHI(:, :, 4)
  W(:, :, 3) = W(:, :, 4)
  OMEGA(:, :, 3) = OMEGA(:, :, 4)
  IF (ctr > 1) THEN
    WRITE(*, '("INITIAL ITERATE TAKEN FROM SOLUTION FOR D =", F7.0)')
    DSTART
  ELSE
    WRITE(*, '("INITIAL ITERATE SET TO ZERO")')
  END IF
ELSE
  ! Otherwise, initialize outer iterates to zero
  PHI(:, :, 3) = 0.0_dp
  W(:, :, 3) = 0.0_dp
  OMEGA(:, :, 3) = 0.0_dp
END IF

!----- Outer iteration loop -----
! Perform nonlinear (outer) iterations until convergence or maximum count.
! ICV is the outer iteration convergence flag: set to 1 if any variable has
not converged in the current outer iteration.

IOUT = 0
IRW = 0
IRO = 0
FAILED = .FALSE.

! Outer iteration loop for nonlinear convergence.
! Repeatedly solve for PHI, W, and OMEGA until all variables converge
or the maximum number of iterations is reached.
OUTER_ITER: DO
  ! Check for maximum number of outer iterations
  IF (IOUT .GE. MAXOUT) THEN
    ! Outer iteration failed: print message and skip to next case
    CALL ERROR_HANDLER(58)
    FAILED = .TRUE.
    EXIT OUTER_ITER
  END IF

  ! Update outer iteration counter and print message
  IOUT = IOUT + 1
  WRITE(*, '("//"OUTER ITERATION", I5//)') IOUT

  ! ICV is the outer iteration convergence flag: set to 1 if any
  variable (PHI, W, OMEGA) has not converged in the current outer
  iteration.
  ! The outer iteration loop exits when ICV == 0, indicating
  convergence.
  ICV = 0

  ! Update previous outer iterates
  PHI(:, :, 1) = PHI(:, :, 3)
  W(:, :, 1) = W(:, :, 3)

```

```
OMEGA(:, :, 1) = OMEGA(:, :, 3)
```

```
! Compute constant SOR coefficient for PHI
```

```
DO I = 2, NR
  DO J = 2, NA
    C(I,J) = B(I,5) * OMEGA(I,J,1)
  END DO
END DO
```

```
!----- SOR for PHI -----
```

```
! Solve for PHI using SOR. If SOR fails to converge, error code 55 is triggered. ↗
```

```
CALL SOR_PHI(PHI, OMEGA, B, C, RHOC, RHO, EPPS, ISOR_PHI, MAXSOR, NR, ↗
  NA, NRP1, NAP1, NRM1, ERROR_HANDLER)
CALL SMOOTH(NRP1, NAP1, PHI, XI(1), XIC(1), EPS(1), ICV)
CALL OUTPUT('PHI ', ISOR_PHI, PHI(1,1,3), NR, NA)
```

```
!----- SOR for W -----
```

```
! Solve for W using SOR. If SOR fails to converge, error code 56 is triggered. ↗
```

```
! If convergence fails, reduce over-relaxation factor and retry up to NOR ↗
times.
```

```
E0 = 4._dp + ABS(PHI(2,NAH,3))
E1 = (1._dp - MIN(PHI(2,NAH,3),0._dp)) / E0
E2 = 2._dp / E0
E3 = (1._dp + MAX(PHI(2,NAH,3),0._dp)) / E0
E4 = DDR2 / E0
```

```
! Compute SOR coefficients for W along boundaries
```

```
DO I = 2, NR
  DELTA1 = DA - PHI(I,2,3)
  DELTA2 = DA + PHI(I,NA,3)
  EE(I) = 2._dp * (DADR + DRDA * RINV2(I))
  EE1 = EE(I) + RINV(I) * ABS(DELTA1)
  EE2 = EE(I) + RINV(I) * ABS(DELTA2)
  E(I,1,1) = (DADR + RINV(I) * MAX(DELTA1,0._dp)) / EE1
  E(I,NAP1,1) = (DADR + RINV(I) * MAX(DELTA2,0._dp)) / EE2
  EF(I) = DRDA * RINV2(I)
  E(I,1,2) = EF(I) / EE1
  E(I,NAP1,2) = EF(I) / EE2
  E(I,1,3) = (DADR - RINV(I) * MIN(DELTA1,0._dp)) / EE1
  E(I,NAP1,3) = (DADR - RINV(I) * MIN(DELTA2,0._dp)) / EE2
  E(I,1,4) = E(I,1,2)
  E(I,NAP1,4) = E(I,NAP1,2)
  E(I,1,5) = DDRDAM / EE1
  E(I,NAP1,5) = DDRDAM / EE2
END DO
```

```
! EE1/EE2: Denominator for SOR coefficients at lower/upper angular ↗
boundaries (alpha=0/pi)
```

```
! These account for upwinding and boundary effects in the SOR update ↗
```

```
for W.
```

```
!----- SOR coefficient setup for W (interior) -----
! Compute SOR coefficients for W at interior points (not on boundaries).
```

```
DO I = 2, NR
  DO J = 2, NA
    GAMMA = .5_dp * (PHI(I+1,J,3) - PHI(I-1,J,3))
    DELTA = DA - .5_dp * (PHI(I,J+1,3) - PHI(I,J-1,3))
    E(I,J,6) = EE(I) + RINV(I) * (ABS(GAMMA) + ABS(DELTA))
    E(I,J,1) = (DADR + RINV(I) * MAX(DELTA,0._dp)) / E(I,J,6)
    E(I,J,2) = (EF(I) + RINV(I) * MAX(GAMMA,0._dp)) / E(I,J,6)
    E(I,J,3) = (DADR - RINV(I) * MIN(DELTA,0._dp)) / E(I,J,6)
    E(I,J,4) = (EF(I) - RINV(I) * MIN(GAMMA,0._dp)) / E(I,J,6)
    E(I,J,5) = DDRDAM / E(I,J,6)
  END DO
END DO
```

```
! Retry SOR for W with reduced over-relaxation factor if convergence fails.
IRW = 0
W_RETRY: DO
  ISOR_W = 0
  CALL SOR_W(W, E, EPPS(2), RHOC(2), RHO(2), ISOR_W, MAXSOR, NR, NA,
    NRP1, NAP1, NAH, E1, E2, E3, E4, DOR, NOR, IRW, ICONV,
    ERROR_HANDLER)
  IF (ICONV .EQ. 0._dp) EXIT W_RETRY
END DO W_RETRY
```

```
!----- Smoothing for W -----
! Smooth W at the origin separately to ensure stability and maintain axisymmetry.
! The rest of the region is smoothed by the SMOOTH subroutine.
```

```
W(1,1,3) = XI(2) * W(1,1,1) + XIC(2) * W(1,1,3)
DO J = 2, NAP1
  W(1,J,3) = W(1,1,3)
END DO
IF (ABS(W(1,1,1) - W(1,1,3)) .GT. EPS(2)) ICV = 1
```

```
! Smooth W in the rest of the region, excluding r = 1
CALL SMOOTH(NRP1, NAP1, W, XI(2), XIC(2), EPS(2), ICV)
CALL OUTPUT('W ', ISOR_W, W(1,1,3), NR, NA)
```

```
!----- SOR for OMEGA -----
! Solve for OMEGA using SOR. If SOR fails to converge, error code 57 is triggered.
! If convergence fails, reduce over-relaxation factor and retry up to NOR times.
```

```
! Apply boundary condition for OMEGA at the outer radial boundary (r = NR).
DO J = 2, NA
```

```

      OMEGA(NRP1,J,3) = XI(3) * OMEGA(NRP1,J,1) - XIC(3) * 2._dp * RINV2
      (2) * PHI(NR,J,3)
      IF (ABS(OMEGA(NRP1,J,1) - OMEGA(NRP1,J,3)) .GT. EPS(3)) ICV = 1
    END DO

!----- Boundary conditions for OMEGA -----
! Apply physical boundary condition for OMEGA at the outer radial boundary
(r=NR).
! This enforces the correct vorticity at the wall based on the current PHI.

      ! Compute constant coefficient for SOR iteration for OMEGA
    DO I = 2, NR
      DO J = 2, NA
        E(I,J,6) = -W(I,J,3) * (SA(J) * (W(I+1,J,3) - W(I-1,J,3)) + CA
          (I,J) * (W(I,J+1,3) - W(I,J-1,3))) / E(I,J,6)
      END DO
    END DO

      ! Retry SOR for OMEGA with reduced over-relaxation factor if
      convergence fails.
      IRO = 0
      OMEGA_RETRY: DO
        ISOR_OMEGA = 0
        CALL SOR_OMEGA(OMEGA, E, RHOC(3), RHO(3), EPPS(3), ISOR_OMEGA,
          MAXSOR, NR, NA, NRP1, NAP1, ICONV, ERROR_HANDLER)
        IF (ICONV .EQ. 0) EXIT OMEGA_RETRY
        IF (ISOR_OMEGA .GE. MAXSOR) THEN
          ! OMEGA iteration failed: reduce over-relaxation factor and try
          again
          CALL ERROR_HANDLER(57, RHO(3))
          IF (IRO >= NOR) EXIT OMEGA_RETRY
          IRO = IRO + 1
          RHO(3) = RHO(3) - DOR(IRO)
          RHOC(3) = 1.0_dp - RHO(3)
          DO I = 1, NRP1
            DO J = 1, NAP1
              OMEGA(I, J, 3) = OMEGA(I, J, 1)
            END DO
          END DO
        END IF
      END DO OMEGA_RETRY

!----- Smoothing for OMEGA -----
! Blend new and previous iterates for OMEGA.

      CALL SMOOTH(NRP1, NAP1, OMEGA, XI(4), XIC(4), EPS(3), ICV)
      CALL OUTPUT('OMEGA', ISOR_OMEGA, OMEGA(1,1,3), NR, NA)

!----- Check for convergence -----
! If all variables have converged (ICV == 0), exit the outer iteration loop.

      IF (ICV == 0) EXIT OUTER_ITER
    END DO OUTER_ITER

```

```

IF (.NOT. FAILED) THEN
  WRITE(*, '("OUTER ITERATION CONVERGED TO GIVEN TOLERANCES.")')
END IF

```

```

!----- Compute flux ratio -----

```

```

! Integrate W over the domain using the trapezoidal rule to compute the
  total flux ratio QR.

```

```

! This provides a measure of the total flow through the tube for the
  current case.

```

```

QR = 0._dp
DO J = 1, NA
  QR = QR + W(1,J,3) + W(2,J,3) + W(2,J+1,3)
END DO
QR = QR * CO(1)
DO I = 2, NR
  S = 0._dp
  DO J = 1, NA
    S = S + W(I,J,3) + W(I+1,J,3) + W(I,J+1,3) + W(I+1,J+1,3)
  END DO
  QR = QR + CO(I) * S
END DO

```

```

! Print out value of flux ratio (QR)
WRITE(*, '("FLUX RATIO =",F10.5)') QR

```

```

!----- Output solution to file if requested -----

```

```

! Write solution arrays and parameters to a file for post-processing.

```

```

IF (IFIL .NE. 0) THEN
  WRITE(file_id, '(i0)') INT(D)
  file_name = 'file_D' // TRIM(ADJUSTL(file_id)) // '.dat'
  PRINT *, "file name is ", file_name
  OPEN(NEWUNIT=unit, FILE=file_name, STATUS='replace')
  WRITE(unit,*) NRP1
  WRITE(unit,*) NAP1
  WRITE(unit,*) XI
  WRITE(unit,*) RHO
  WRITE(unit,*) EPS
  WRITE(unit,*) D
  WRITE(unit,*) QR
  DO I = 1, NRP1
    WRITE(unit,*) PHI(I,:,3)
  END DO
  DO I = 1, NRP1
    WRITE(unit,*) W(I,:,3)
  END DO
  DO I = 1, NRP1
    WRITE(unit,*) OMEGA(I,:,3)
  END DO
  CLOSE(unit)
END IF

```

```
      IF (IFIL .NE. 0) WRITE(*,('SOLUTION WAS OUTPUT TO FILE.))

!----- Save solution in memory for next case -----
! Store current solution as initial guess for next D.

      PHI(:, :, 4) = PHI(:, :, 3)
      W(:, :, 4) = W(:, :, 3)
      OMEGA(:, :, 4) = OMEGA(:, :, 3)

      DSTART = D

!----- End of main case loop -----

      IF (ISAVE /= 0) THEN
        PHI(:, :, 4) = PHI(:, :, 3)
        W(:, :, 4) = W(:, :, 3)
        OMEGA(:, :, 4) = OMEGA(:, :, 3)
        DSTART = D
      END IF

      PRINT *, 'Done with case D =', D
    END DO

!----- Timing and program end -----
! Print elapsed CPU time and prompt user to exit.

    CALL SYSTEM_CLOCK(COUNT=clock_end)
    elapsed_time = REAL(clock_end - clock_start, KIND=dp) / REAL(clock_rate,
      KIND=dp)
    PRINT *, '[SYSTEM_CLOCK] Elapsed CPU time (seconds):', elapsed_time

    PRINT *, 'D-loop complete: To exit, press <ENTER>'
    READ(*,*)

  END PROGRAM MAIN
```