User's guide and program description of ACOUSTIC_RANGING

Technische
Universiteit
Eindhoven

Instituut
Wiskundige Dienstverlening
Eindhoven

Report 92-08

User's guide and Program Description of

ACOUSTIC_RANGING

A.F.M. Baaijens

August 1992

Den Dolech 2
Postbus 513
5600 MB Eindhoven

Report 92-08

User's guide and Program Description of

ACOUSTIC_RANGING

A.P.M. Baaijens

August 1992

Summary.

This report describes the program ACOUSTIC_RANGING and its use.
ACOUSTIC_RANGING is a program to analyse by a mathematical model the reflection and transmission properties of acoustic waves in a coal gasification exhaust pipe, including the effects of temperature gradients and the pipe exit. The mathematical background is given in [1]. The numerical analysis is carried out in cooperation with Prof. R.M.M. Mattheij.

This investigation has been performed under contract with the "NV tot Keuring van Elektrotechnische Materialen" (KEMA), under contract WDC 138.

The program acoustic_ranging

13 August 1992. SUN version.

INTRODUCTION

1. Calling the program

This will be done with the command:

```
acoustic_ranging input_name output_name
```

where acoustic_ranging is the executable file and the two arguments are the names of the input and output file.

SUBPROGRAM MAIN

1. Parameters and variables

Only the relevant parameters and variables will be described.

a	double precision. duct radius.
alfa	double precision array of mmm. alfa(i)=zjl(i)/a
consta	double precision. constant used when scaling the modes.
ckk	double precision. constant used in the function kk.
D	double precision. position of the diaphragm.
dmatW	double precision array of dimension mmm contains the diagonal elements of the diagonal matrix M.
epsint	double precision. accuracy integration routine.
f	double precision. frequency.
gamma	double precision. gas constant.
gammaD	double complex array of dimension mmm. contains the (mumax) eigenvalues at x=D.
gammaL	double complex array of dimension mmm. contains the (mumax) eigenvalues at x=L.
gamma0	double complex array of dimension mmm. contains the (mumax) eigenvalues at x=0.
gammdak	double complex array of dimension mmm. contains the (nhs) eigenvalues at x=D.
gammux	double complex array of dimensions (mmm,nxm). contains the eigenvalues of the eigenvalue problem for m=0 (axial wave numbers). gammux(i,j) contains the i-th eigenvalue matching x(j), j=1,.....,nx.
gammu1	double complex array of dimensions (mmm,nxm). contains the eigenvalues of the eigenvalue problem for m=1 (axial wave numbers). gammu1(i,j) contains the i-th eigenvalue matching x(j), j=1,.....,nx.
kkwad	double precision array of dimension n+1. contains squares of k(x,r) values.

k0	double precision. k(x=L+)
L	double precision. duct length.
matD	double complex array of dimensions (mmm,mmm). contains the values of the elements of the matrix D.
matE	double complex array of dimension mmm. contains the values of the diagonal elements of the diagonal matrix E.
matF	double complex array of dimension mmm. contains the values of the diagonal elements of the diagonal matrix F.
matG	double precision array of dimensions (mmm,mmm). contains the values of the elements of the matrix G.
matJ	double precision array of dimension mmm. matJ(i)=J0(zj1(i)).
matJ0	double precision array od dimensions (n+1,mmm). matJ0(i,j)=JB0(alfa(j)*(i-1)*step).
matM	double complex array of dimensions (mmm,mmm). contains the values of the elements of the matrix M.
matR	double complex array of dimensions (mmm,mmm). contains the values of the elements of the matrix R. reflection matrix (obstacle).
matRE	double complex array of dimensions (mmm,mmm). contains the values of the elements of the matrix R_E. reflection matrix (end).
matS	double complex array of dimensions (mmm,mmm). contains the values of the elements of the matrix S.
matT	double complex array of dimensions (mmm,mmm). contains the values of the elements of the matrix T.
maxmu	integer. maximum even number of modes that is output.
mmm	integer. upperbound for mumax and default value.
mumax	integer. number of modes in the duct.
n	integer. n+1 is the odd number of points on the r-grid of the interval [0,a].
nDstep	integer. even number of steps from 0 to D on the x-grid.
Nh	integer number of modes on the interval [0,h]. Nh=(h/a)*mumax.
nhs	integer. nhs+1 is the odd number of points on the r-grid of the interval [0,h].
nLstep	integer. even number of steps from D to L on the x-grid.
nx	integer. nx=nDstep+nLstep+1.
nxm	integer. upperbound for nx.
n0,n1	double precision. parameters defining the temperature profile.
only	logical. if true only eigenvalues will be output.
outnr	integer. number of output points of the r-grid.
pin,pref,ptot	double complex arrays of dimension n+1. contains the pressure profiles over the r-grid at x=0.
progress	logical. if true messages of the progress will be sent to the display when the programm is running.
psiD	double precision array of dimensions (n+1,mmm). contains the modes for x=D.
psidak	double precision array of dimensions (n+1,mmm).

	contains the modes for x=D on the interval [0,h].
psiL	double precision array of dimensions (n+1,mmm).
	contains the modes for x=L.
psi0	double precision array of dimensions (n+1,mmm).
	contains the modes for x=0.
power	double precision.
	contains the value of P.
power0	double precision.
	contains the value of P0.
R	double precision.
	gas constant.
step	double precision.
	width of an interval on the r-grid (= a/n).
Tchoice	integer.
	will be used to choose a temperature profile.
Tend	double precision.
	temperature at the duct end x=L+.
T0,T1	double precision.
	parameters defining the temperature profile.
veca	double complex array of dimension mmm.
	incident amplitude vector.
vecaD	double complex array of dimension mmm.
	amplitude vector A(D).
vecb	double complex array of dimension mmm.
	reflection vector B (with obstacle).
vecbD	double complex array of dimension mmm.
	reflection vector B(D).
vecb0	double complex array of dimension mmm.
	reflection vector B0 (without obstacle).
veca	double complex array of dimension mmm.
	incident amplitude vector.
x	double precision array of dimension nxm.
	contains the points of the xgrid.
zj1	double precision array of dimension mmm.
	contains in ascending order the zeroes of the
	derivative of the Bessel function J0.
zj2	double precision array of dimension mmm.
	zj2(i)=zj1(i)**2.

2. Auxilliary Routines.

The program calls the following routines:

the BLAS routines	ZGEMM and ZGEMV,
the LINPACK routines	ZGECO and ZGESL,
the 'home-made' routines	BSNULP, FUNT, GAMSLM, GAMSOL, INPROD, INTEGR, JB0 and KK.

The BLAS and LINPACK routines are obtained from NETLIB with the internet address "NETLIB@RESEARCH.att.com".
The documentation of these routines will be found in the listings.
The other routines are documented here.

3. description

The main program can be divided in an number of consecutive blocks each with its own function.
Every block begins with a label of the form 88xx.
The description, given here makes use of these labels.

label 8800	The input is read by a namelist READ statement with the group name PRMTRS.
label 8805	Default values for the input parameters are set.

label 8810 The names of input and output file are read, the files are opened, the input file is read and the correctness of the input is checked.

label 8815 Computation of program constants.

label 8820 The input data and some program constants are written to the output file. WARNING: the parameters nDstep, nLstep, mumax and h may be changed (adjusted) in case of incorrect input.

label 8825 Computation of the axial wave numbers for m=0 and m=1. They are stored in the arrays gammux and gammul respectively. Then they are written to the output file.

label 8830 Computation of the axial wave numbers and modes for x = 0, D and L. They are stored in the arrays gamma0, gammaD, gammaL, psi0, psiD and psiL.

label 8835 Computation of the axial wave numbers and modes for x=D on the interval [0,h]. They are stored in the arrays gamdak and psidak.

label 8840 Computation of the matrices M(D) and S(D) which are stored in the arrays matM and matS.

label 8845 Computation of the transmission matrix T which is stored in the array matT.

label 8850 Computation of the diagonal matrices M, E and F and the reflection matrix R which are stored in the arrays dmatW, matE, matF and matR.

label 8855 Computation of the diagonal matrix J and the matrices G a n d G*(inverse of J) which are stored in the arrays matJ, matG and GJmin1.

label 8860 Computation of the matrices D and K which are stored in the arrays matD and matK.

label 8865 Computation of the reflection matrix R_E which is stored i n the array matRE.

label 8870 Computation of the vectors A(D), B(D), B and B0 which are stored in the arrays vecaD, vecbD, vecb and vecb0.

label 8875 Computation of the powers P and P0 which are stored in the variables P and P0.

label 8880 Computation of the pressure distribution p_in, p_ref and p_tot which are stored in the arrays pin, pref and ptot.

label 8885 Writing of the results to the output file.

SUBROUTINE RWPSOL

1. Purpose

The routine solves the eigenvalue problem described in [1] for m = 0.

2. Specification

```
SUBROUTINE RWPSOL(X,H,N,CONSTA,IERR,MUMAX,GAMMA,PSI,LDAPSI)
INTEGER N,IERR,MUMAX,LDAPSI
DOUBLE PRECISION X,H,CONSTA,PSI(LDAPSI,MUMAX)
DOUBLE COMPLEX GAMMA(MUMAX)
```

3. Description

By calling the subroutine EIGEN twice two approximations of the solution of the eigenvalue problem are computed. By means of extrapolation an improved approximation will be obtained.

4. Parameters

X	Point on the x-grid.
H	inner radius of the diaphragm.
N	N+1 = number of points on the r-grid.
CONSTA	scaling constant for psi.
IERR	error indicator on exit IERR contains 0 unless an error has occurred.

MUMAX	number of modes in the duct.
GAMMA	contains on exit the eigenvalues.
PSI	contains on exit the modes.
LDAPSI	leading dimension of the array PSI as declared in the MAIN.

5. Auxilliary Routines

This routine calls routine EIGEN.

SUBROUTINE EIGEN
==========

1. Purpose

This routine is an auxilliary routine which will be called by the routine RWPSOL.

2. Specification

```
SUBROUTINE EIGEN(X,A,N,CONSTA,IERR,MUMAX,GAMMA2,PSI)
INTEGER N,NM,IERR,MUMAX
DOUBLE PRECISION X,A,CONSTA,GAMMA2(MUMAX),PSI(NM,MUMAX)
```

3. Description

EIGEN computes the modes and squares of the gamma's (eigenvalues) of the eigenvalue problem for m = 0 and h = a.

4. Parameters

A	duct radius or inner radius of the diaphragm.
NM	leading dimension of the arrays PSI1 and PSI2 as declared i n RWPSOL.
GAMMA2	contains on exit the squares of the gamma's (eigenvalues).

The meaning of the rest of the parameters is analogous to that of the the parameters from the routine RWPSOL.

5. Auxilliary Routines

This routine calls the routines TSTURM and the function KK. TSTURM is a EISPACK routine for finding the eigenvalues and associated eigenvectors of a tridiagonal symmetric matrix. KK computes the function value k(x,r).

FUNCTION FASE
==========

1. Purpose

The function computes the phase of a complex number in degrees.

2. Specification

```
DOUBLE PRECISION FUNCTION FASE(P)
DOUBLE COMPLEX P
```

FUNCTION FUNT
==========

1. Purpose

The function computes the temperature in the duct.

2. Specification

```
   DOUBLE PRECISION FUNCTION FUNT(X,R)
   DOUBLE PRECISION X, R
```

3. Parameters

X	point on the x-grid.
R	point on the r-grid.

The function uses the COMMON BLOCK TCOM which defines the temperature profile.

SUBROUTINE GAMSOL

1. Purpose

The routine computes the eigenvalues gamma of the eigenvalue problem for m = 0 described in [1].

2. Specification

```
   SUBROUTINE GAMSOL(X,H,N,IERR,MUMAX,GAMMA)
   INTEGER N,IERR,MUMAX
   DOUBLE COMPLEX GAMMA(MUMAX)
```

3. Description

By calling the subroutine EIGAM twice two approximations of the eigenvalues are computed.
By means of extrapolation an improved approximation will be obtained.

4. Parameters

The meaning of the parameters is analogous to that of the parameters from the subroutine RWPSOL.

5. Auxilliary routine

This routine calls the routine EIGAM.

SUBROUTINE EIGAM

1. Purpose

This routine is an auxilliary routine which will be called by the routine GAMSOL.

2. Specification

```
   SUBROUTINE EIGAM(X,A,N,IERR,MUMAX,GAMMA2)
   INTEGER N,IERR,MUMAX
   DOUBLE PRECISION X,A,GAMMA2(MUMAX)
```

3. Description

EIGAM computes the squares of the gamma's (eigenvalues) of the eigenvalue problem for m = 0 and h = a.

4. Parameters

The meaning of the parameters is analogous to that of the parameters from the routine EIGEN.

5. Auxilliary Routines

This routine calls the routines GSTURM and the function KK.
GSTURM ia a stripped version of the EISPACK routine TSTURM for finding the eigenvalues and associated eigenvectors of a tridiagonal symmetric matrix.
KK computes the function value k(x,r).

SUBROUTINE GAMSLM

1. Purpose

The routine computes the eigenvalues gamma of the eigenvalue problem for m > 0 described in [1].

2. Specification

```
SUBROUTINE GAMSLM(EM,X,H,N,IERR,MUMAX,GAMMA)
INTEGER EM,N,IERR,MUMAX
DOUBLE COMPLEX GAMMA(MUMAX)
```

3. Description

By calling the subroutine EIGAMM twice two approximations of the eigenvalues are computed.
By means of extrapolation an improved approximation will be obtained.

4. Parameters

EM parameter m from the eigenvalue problem.

The meaning of the other parameters is analogous to that of the parameters from the subroutine GAMSOL.

5. Auxilliary routine

This routine calls the routine EIGAMM.

SUBROUTINE EIGAMM

1. Purpose

This routine is an auxilliary routine which will be called by the routine GAMSLM.

2. Specification

```
SUBROUTINE EIGAMM(EM,X,A,N,IERR,MUMAX,GAMMA2)
INTEGER EM,N,IERR,MUMAX
DOUBLE PRECISION X,A,GAMMA2(MUMAX)
```

3. Description

EIGAMM computes the squares of the gamma's (eigenvalues) of the eigenvalue problem for m > 0 and h = a.

4. Parameters

EM the parameter m from the eigenvalue problem.

The meaning of the other parameters is analogous to that of the parameters from the routine EIGAM.

5. Auxilliary Routines

This routine calls the routines GSTURM and the function KK.
GSTURM ia a stripped version of the EISPACK routine TSTURM for finding the eigenvalues and associated eigenvectors of a tridiagonal symmetric matrix.
KK computes the function value k(x,r).

SUBROUTINE EORF

1. Purpose

The routine computes the diagonal elements of the diagonal matrices E or F described in [1].

2. Specification

```
SUBROUTINE EORF(LBJ,UBJ,MUMAX,STEP,GAMMUX,MMM,,NXM,MATE)
INTEGER LBJ,UBJ,MUMAX,MMM,NXM
DOUBLE PRECISION STEP
DOUBLE COMPLEX GAMMUX(MMM,NXM),MATE(MUMAX)
```

3. Description

The integrals that appear in the formulas for the diagonal elements of E and F will be aproximated by Simpson's composite integration rule.

4. Parameters

LBJ,UBJ	lower- and upperbound of the index j that points to the j-th integration subinterval.
MUMAX	order of the matrix.
STEP	integration step (=a/n).
GAMMUX(I,J)	i-th eigenvalue matching x(j).
MMM	leading dimension of GAMMUX as declared in the MAIN.
NXM	second dimension of GAMMUX as declared in the MAIN.
MATE	on exit contains the values of the diagonal elements.

FUNCTION INPROD

1. Purpose

INPROD approximates the integral of psi_mu(r)*psi_nu(r)*r/k(x,r)**2 as function of r from 0 to a or h.

2. Specification

```
DOUBLE PRECISION FUNCTION INPROD(NHS,STEP,PSI1,PSI2,KKWAD)
INTEGER NHS
DOUBLE PRECISION STEP,PSI1(NHS+1),PSI2(NHS+1),KKWAD(NHS+1)
```

3. Description

INPROD computes an approximation of the integral of psi1(r)*psi2(r)*r/k(x,r)**2 as function of r from 0 to nhs*step, by means of the composite formula of Simpson.

4. Parameters

NHS	even number of integration subintervals.
STEP	integration step (= a/n).

PSI1,PSI2 modes (eigenvectors).
KKWAD contains the squares of the k(x,r) values.

FUNCTION INTEGR

1. Purpose

The function calculates the integral defining the elements of the matrix matD.

2. Specification

```
   COMPLEX*16 FUNCTION INTEGR()
```

3.Description

The integral is split up in 4 parts: 2 real integrals I1 and I2 (evaluated by Rombergs trapezium method ROMBERG with INTGND1 and INTGRND2), the complex integral I3 (evaluated by CROMBERG with INTGRND3), and a residue contribution if MU=NU.
Each integrand consists of an expensive MU,NU-independent factor and an inexpensive MU,NU-dependent factor. Each expensive function factor evaluation is executed only once and then saved for later use in 3 arrays IK1,IK2,IK3 (of size IK1TAB,IK2TAB,IK3TAB) in the same order as calculated.
The number of initialised elements of IKi is counted TELMAX(i).
If TELMAX(i) is ever to become larger than IKiTAB, the IKiTAB is to be set to a higher number (2**k + 1).
TELMAX is set to zero before the first call to INTEGR, and passed to INTGRND1,2,3 via COMMON//.

4. Parameters

EPSINT	integration accuracy
TELMAX(3)	maximum index thus far of arrays with MU,NU-independent integrand factor.
EERSTE	logical, =TRUE if counter in MU,NU-independent integrand factor is to be reset to 1.
A,B	ends of integration interval
MAXITER	maximum number of Romberg iterations (default=10)
INTGRND1 \	externals of
INTGRND2 >	(C)ROMBERG calls;
INTGRND3 /	denote the integrand functions.
I1,I2,I3	integrals: results of (C)ROMBERG.

5. Auxiliary routines

The function calls ROMBERG and CROMBERG, which call INTGRND1,2,3 , and these call the complex Bessel function routines KB, IB, and H2B.

FUNCTION JB0

1. Purpose

The function computes the real Bessel function of the first kind of order 0 for real argument.

2. Specification

```
   REAL*8 FUNCTION JB0(X)
   REAL*8 X
```

3. Auxilliary Routine

The function calls the more general Bessel function routine JB.

FUNCTION KK

1. Purpose

The function computes the local wave number k(r,x).

2. Specification

```
DOUBLE PRECISION FUNCTION KK(R,X)
DOUBLE PRECISION R, X
```

3. Parameters

R point on the r-grid.
X point on the x-grid.

The function uses the COMMON BLOCK KKCOM.

4. Auxilliary function

The function calls the function FUNT.

1. S.W. Rienstra, "A Multiple Scales, Modal Expansion Solution for the Acoustical Detection of Obstructions in a Coal Gasification Exhaust Pipe",
Report IWDE 92-07, August 1992

Program Structure Diagram for the program acoustic_ranging.

Set default values for the input parameters.

Open input file.

Open output file.

Read input parameters.

Change the value of the parameters nDstep, nLstep and mumax in the event of incorrect input and print a warning for any change.

Compute program constants.

Adjust h if h/step is not an even integer number.

Write the input data and some program constants to the output file.

Compute the axial wave numbers for m=0 and write these to the output file.

Compute the axial wave numbers for m=1 and write these to the output file.

only axial wave numbers ?

no

yes

Compute the axial wave numbers and modes for x = 0, D and L.

h = a ?

yes

no

Transmission matrix

T := I

Compute the axial wave numbers and modes for x = D on the interval [0,h].

Compute the matrix M(D).

Compute the matrix S(D).

Compute the transmission matrix T.

Compute the reflection matrix R = I - T .

Compute the diagonal matrix J.

Compute the diagonal matrix W.

Compute the diagonal matrix E.

Compute the diagonal matrix F.

Compute the matrix G and G * (inverse of J) .

Compute the matrix D.
Compute the matrix K.
Compute the matrix R_E.
Compute the vector A(D).
Compute the vector B(D).
Compute the vectors b and b_0.
Compute the power P.
Compute the power P_0.
Compute the pressure distributions p_in, p_ref and p_tot.
Write consecutively to the output file : the reflection vectors b and b_0, the powers P, P_0 and 10*log10(P_0/P), the reflection matrix R, the reflection matrix R_E, modulus and phase of p_in, p_ref and p_tot, the distribution of the temperature T.

```
C     program acoustic_ranging
C     ------------------------
C     SUN version  31 August  '92
C
C     input parameters
C     ----------------
C     f               frequency
C     mumax           maximum number of modes
C     veca            incident field (vector A_mu)
C     L               duct length
C     a               duct radius
C     D               position of the diaphragm
C     h               inner radius of the diaphragma
C     p0              atmospheric pressure
C     gamma           gas constant
C     R               gas constant
C     nDstep          even number of steps from 0 to D on the x-grid
C     nLstep          even number of steps from D to L on the x-grid
C     only            if only then only gamma_mu(x) is output
C     maxmu           maximum even number of modes that is output
C     outnr           (appox.) number of output points of the r-grid
C     Tchoice         to choose temperature profile
C     funT(x,r)       defines the temperature profile
C     T0,T1,n0,n1     parameters defining temperature profile
C     Tend            temperature at the duct end x=L+
C     outnx           (approx.) number of output points of the x-grid
C     n               n+1 is the (odd) number of points on the r-grid.
C     progress        logical , if true then information of the progress
C                     of computations will be displayed on the screen.
C
C
C     program variables
C     -----------------
C     Nh      (=mumax*h/a) number of modes in the diagphragm
C             (rows of the matrix S)
C             Note: adjust mumax,h and a for Nh large enough.
C     mumax   number of modes in the duct
C             (columns of the matrix S)
C     mmm     (=30) upperbound for mumax and default value of mumax.
C     nn      (=300) upperbound for n.
C     nhs     (h/a)*n must be an even integer
C             if necessary h will be adjusted to satisfy this condition.
C     step    a/n, the mesh width of the r-grid.
C     consta  constant used when scaling the columns of the matrix psi
C     nx      nDstep+nLstep+1
C     nxm     (=101) odd upperbound for nx.
C     x(j)            j-th interval point on the x-grid.
C     gammux(mu,j)    mu-th axial wave number gamma_mu(x_j)
C                     at j=1,...,nx with m=0
C     gammu1(mu,j)    mu-th axial wave number gamma_mu(x_j)
C                     at j=1,...,nx with m=1
C     Lcopy   L
C     k0      = k(x=L+)
C
      implicit NONE
C
      integer nn,n,ierr,i,nhs,Nh,j,nDstep,mumax, nLstep,
     +        mu, nu, telmax(3), nx, maxmu, mmm, colnr1,
     +        colnr2,outnr,dnx,dnr,indx,nxm,outnx,Tchoice
C
      parameter(mmm=30,nn=300, nxm=101)
C
      logical eerste, only, progress
C
      integer ipvt(mmm), iargc
      double precision a,psi0(nn+1,mmm),psiD(nn+1,mmm),psidak(nn+1,mmm),
     +                 h, kkwad(nn+1), help, rcond, inprod, step, kk,
     +                 L, D, epsint, zj1(mmm), psiL(nn+1,mmm), consta,
     +                 zj2(mmm),matJ(mmm), k0, pi, k0a, JB0, hulp1(nn+1),
     +                 alfa(mmm),matJ0(nn+1,mmm),matG(mmm,mmm),gamma,p0,
     +                 dmatW(mmm), sigma1, sigma2, power, power0,
```



```
     +                 Tend, T0, T1, n0, n1, ckk, f, funT, hulp2(nn+1),
     +                 R, Dstep, Lstep , x(nxm), Lcopy, fase
      double complex gamma0(mmm),gammaD(mmm),gamdak(mmm),iu,
     +       matM(mmm,mmm), mathlp(mmm,mmm),
     +       matS(mmm,mmm),matSM(mmm,mmm),gammaL(mmm),vecaD(mmm),
     +       matU(mmm,mmm), matT(mmm,mmm), z(mmm), vecbD(mmm),
     +       noemer,matE(mmm),matF(mmm),vecb0(mmm), z1(mmm),
     +       matR(mmm,mmm),matD(mmm,mmm),integr,gammu1(mmm,nxm),
     +       GJmin1(mmm,mmm), matK(mmm,mmm), alpha, beta,
     +       KplusI(mmm,mmm),matRE(mmm,mmm), IminRL(mmm,mmm),
     +       matL(mmm,mmm), veca(mmm),vecb(mmm),vecc(mmm),
     +       gammux(mmm,nxm),pin(nn+1), pref(nn+1), ptot(nn+1)
C
      COMMON // epsint,mu,nu, k0a, zj1, zj2, matJ, eerste, telmax, pi
      COMMON /Tcom/ T0, T1, a, n0, n1, L, Tchoice
      COMMON /kkcom/ ckk, k0, Lcopy
C
      character*25 invoer, uitvoer
C
      external RWPSOL, inprod , integr, zgemm, zgemv, zgeco, zgesl,
     +         gamsol, gamslm, JB0, bsnulp, kk, funT
C
8800  NAMELIST /PRMTRS/ f, mumax, veca, L, a, D, h, p0, gamma, R,
     +                  nDstep, nLstep, only, maxmu, n,outnr, n0, n1,
     +                  T0, T1, Tend, outnx,Tchoice,progress
C
C     Default values
C     --------------
8805  consta=1.0d0
      f=100.0d0
      h=0.3d0
      L=10.d0
      a=0.75d0
      D=8.0d0
      mumax=mmm
      do 1 mu=1, mmm
          veca(mu)=dcmplx(0.0d0)
1     continue
      p0=101325d0
      gamma=1.402d0
      R=287d0
      only=.false.
      maxmu=5
      outnr=5
      outnx=3
      nDstep=2
      nLstep=2
      Tchoice=1
      n0=12.0d0
      n1=2.0d0
      T0=1250.0d0
      T1=500.0d0
      Tend=1250.0d0
      progress=.true.



C     SUN FORTAN statements for reading the names of input and output
C     files from the command line.

      if(iargc().ne.2) then
        print *, 'error in command line'
        stop
      endif
      call getarg(1,invoer)
      call getarg(2,uitvoer)
C
```

```
C     Opening of I/O files and reading of the input
C
8810  open(unit=8, file=invoer, status='old')
      open(unit=9, file=uitvoer, status='unknown')
      read(unit=8, nml=PRMTRS, err=2)
      goto 3
 2    write(*, '("error in input file")')
      stop
 3    if(mod(nDstep,2).ne.0) then
        write(*, '("WARNING: nDstep not even and will be adjusted")')
        nDstep=nDstep+1
      endif
      if(mod(nLstep,2).ne.0) then
        write(*, '("WARNING: nLstep not even and will be adjusted")')
        nLstep=nLstep+1
      endif
      if(mumax.gt.mmm) then
        write(*, '("WARNING: mumax greater than mmm and will be",
     +           " adjusted")')
        mumax=mmm
      endif
C

C     computation of program constants
C     --------------------------------
8815  iu=(0.0d0,1.0d0)
      pi=acos(-1d0)
      ckk=2.0d0*pi*f/sqrt(gamma*R)
      k0=ckk/sqrt(Tend)
      Lcopy=L
      epsint=1.0d-8
      k0a=k0*a
      step=a/n
      nx=nDstep+nLstep+1
      if(nx.gt.nxm) then
        write(*, '("WARNING: nDstep+nLstep+1 > nxm and will be",
     +           " adjusted")')
        nDstep=(nxm-1)/2
        nLstep=(nxm-1)/2
        nx=nxm
      endif
      if(outnr.le.1) outnr=2
      if(outnr.gt.n+1) outnr=n+1
      dnr=n/(outnr-1)
      if(outnx.le.2) outnx=3
      if(outnx.gt.nx) outnx=nx
      dnx=(nx+2)/outnx
C
C     h will be adjusted if h/step is not an even
C     integer number.
C
      nhs=2*nint(h/(2*step)-1.d-8)
C     The nearest integer is not uniquely defined in FORTAN for 0.5
C
      h=nhs*step
C

8820  write(9, '("Output from program acoustic_ranging"/
     +         "********************************"//
     +          "Input data"/"----------"/)')
      write(9, '("frequency                          =", d18.10/
     +         "mumax                                =", i3/)')
     +         f, mumax
      write(9, '("incident field")')
      write(9, '(2d18.10)') (veca(j), j=1, mumax)
      write(9, '(/)')
      write(9, '("duct length L                      =", d18.10/
     +         "duct radius a                        =", d18.10/
     +         "diaphragm position D                 =", d18.10/
     +         "diaphragm inner radius h             =", d18.10/
     +         "atmospheric pressure p0              =", d18.10/
```



```
     +         "gas constant gamma                   =", d18.10/
     +         "gas constant R                       =", d18.10/
     +         "nDstep                               =", i3/
     +         "nLstep                               =", i3/
     +         "n0                                   =", d18.10/
     +         "n1                                   =", d18.10/
     +         "T0                                   =", d18.10/
     +         "T1                                   =", d18.10/
     +         "Tend                                 =", d18.10/
     +         "Tchoice                              =", i3/
     +         "k0a                                  =", d18.10/
     +         "number of points of the r-grid       =", i3/)')
     +         L,a,D,h,p0,gamma,R,nDstep, nLstep,n0,n1,
     +         T0,T1,Tend,Tchoice,k0a, n+1

C
C     Computation of gamma_mu(x_j) , mu=1,..,mumax, j=1,....,nx.
C     ----------------------------------------------------------
C     m = 0, 1
C     --------
C
8825  if(progress) then
        write(*, '("Computation of eigenvalues gamma_mu(x_j)")')
        write(*, '("j=1,...,",i2)') nx
      endif
      Dstep=D/nDstep
      Lstep=(L-D)/nLstep
      do 310 j=1, nDstep +1
         x(j)=(j-1)*Dstep
         call gamsol(x(j), a, n, ierr, mumax, z)
         call gamslm(1, x(j), a, n, ierr, mumax, z1)
         if(ierr.ne.0) then
            write(*,'(
     +           "ierr not equal zero on exit subroutine gamslm")')
            stop
         endif
         do 305 i=1, mumax
            gammux(i,j)=z(i)
            gammul(i,j)=z1(i)
305      continue
         if(progress) then
           write(*, '("j=", i2)') j
         endif
310   continue
C
      do 320 j=nDstep+2, nx
         x(j)=D+(j-nDstep-1)*Lstep
         call gamsol(x(j), a, n, ierr, mumax, z)
         call gamslm(1, x(j), a, n, ierr, mumax, z1)
         if(ierr.ne.0) then
            write(*,'(
     +           "ierr not equal zero on exit subroutine gamslm")')
            stop
         endif
         do 315 i=1, mumax
            gammux(i,j)=z(i)
            gammul(i,j)=z1(i)
315      continue
         if(progress) then
           write(*, '("j=", i2)') j
         endif
320   continue
      write(9,'(//"Eigenvalues gamma_mu0(x) as function of x"/
     +         "-------------------------------------"//
     +         "  x                        mu=1")')
      do 635 j=1,nx, dnx
         write(9,'(d18.10,4x,2d18.10)') x(j),gammux(1,j)
635   continue
      write(9,'(//)')
      colnr2=1
640   colnr1=colnr2+1
```

```
      colnr2=colnr2+2
      if(colnr2.gt.maxmu) colnr2=maxmu
      if(colnr1.eq.colnr2) then
         write(9, '(i3)') colnr1
      else
         write(9,'(i3,38x,i2)') colnr1, colnr2
      endif
      do 650 i=1, nx, dnx
       write(9,'(2d18.10,4x,2d18.10)') (gammux(mu,i),mu=colnr1,colnr2)
650   continue
      write(9,'(/)')
      if(colnr2.ne.maxmu) goto 640

      write(9,'(//"Eigenvalues gamma_mu1(x) as function of x"/
     +          "---------------------------------------"//
     +        "  x                    mu=1")')
      do 636 j=1,nx, dnx
         write(9,'(d18.10,4x,2d18.10)') x(j),gammu1(1,j)
636   continue
      write(9,'(//)')
      colnr2=1
641   colnr1=colnr2+1
      colnr2=colnr2+2
      if(colnr2.gt.maxmu) colnr2=maxmu
      if(colnr1.eq.colnr2) then
         write(9, '(i3)') colnr1
      else
         write(9,'(i3,38x,i2)') colnr1, colnr2
      endif
      do 651 i=1, nx, dnx
       write(9,'(2d18.10,4x,2d18.10)') (gammu1(mu,i),mu=colnr1,colnr2)
651   continue
      write(9,'(/)')
      if(colnr2.ne.maxmu) goto 641
      if(only) GOTO 9999
C

C     Computation of psi(mu) and gamma(mu) , mu=1,2,.....,mumax
C     ---------------------------------------------------------
C     for x = 0,D,L
C     -------------
C
8830  if(progress) then
         print *, 'Computation of psi(x=0)'
      endif
      call RWPSOL(0.0d0,a,n,consta,ierr,mumax,gamma0,psi0,nn+1)
C     write(*, '(2d18.10)') (gamma0(i), i=1, mumax)
      if(ierr.ne.0) then
         write(*, '("No solution psi(0) in RWPSOL; ierr=",i2)') ierr
         stop
      endif
      if(progress) then
         print *, 'Computation of psi(x=D)'
      endif
      call RWPSOL(D, a, n, consta, ierr, mumax, gammaD, psiD,nn+1)
C     write(*, '(2d18.10)') (gammaD(i), i=1, mumax)
      if(ierr.ne.0) then
         write(*, '("No solution psi(D) in RWPSOL; ierr=",i2)') ierr
         stop
      endif
      if(progress) print *, 'Computation of psi(x=L)'
      call RWPSOL(L, a, n, consta, ierr, mumax, gammaL, psiL,nn+1)
C     write(*, '(2d18.10)') (gammaL(i), i=1, mumax)
      if(ierr.ne.0) then
         write(*, '("No solution psi(L) in RWPSOL; ierr=",i2)') ierr
         stop
      endif
C
C
C Computation of matrices
C -----------------------
```



```
C
      if(h.eq.a) then
        do 10 j=1, mumax
           do 15 i=1, mumax
              matT(i,j)=dcmplx(0.0d0,0.0d0)
15         continue
           matT(j,j)=dcmplx(1.0d0,0.0d0)
10      continue
      else
           Nh=nint(mumax*h/a-1.d-8)
C  Afronden van 0.5 niet eenduidig in Fortran gedefinieerd!
           if(Nh.lt.4) then
              write(*, '("Terminal Error : "/
     +             " Too small number of modes in the diaphragm")')
              stop
           endif
           do 20 i=1,n+1
              kkwad(i)=kk((i-1)*step,D)**2
20         continue


8835       call RWPSOL(D,h,nhs,consta,ierr,Nh,gamdak,psidak,nn+1)
C     write(9, *) D, h, nhs, consta, step
C     write(9, '(//d18.10)') (psidak(i,1), i=1, 25)
C     stop
           if(ierr.ne.0) then
             write(*,'("No solution psidak in RWPSOL; ierr=",i2)')
     +             ierr
                   stop
           endif
C

C     Computation of the matrices M, S and S*M
C     ----------------------------------------
C
8840  do 30 i=1, mumax
         do 32 indx=1, n+1
              hulp1(indx)=psiD(indx,i)
32       continue
         if(dimag(gammaD(i)).eq.0.0d0) then
              noemer=dcmplx(consta)
         else
              noemer=-iu*consta
         endif
         do 40 j=1, Nh
            do 42 indx=1, n+1
              hulp2(indx)=psidak(indx,j)
42          continue
            help=inprod(nhs, step, hulp1, hulp2, kkwad)
            matM(i,j)=help/noemer
40       continue
30    continue
      do 50 i=1, Nh
         do 52 indx=1, n+1
            hulp1(indx)=psidak(indx,i)
52       continue
         do 60 j=1, mumax
            do 62 indx=1, n+1
              hulp2(indx)=psiD(indx,j)
62          continue
            help=inprod(nhs, step, hulp1, hulp2, kkwad)
            matS(i,j)=dcmplx(help*abs(gamdak(i)))
60       continue
50    continue
C
C

C     Computation of the matrix U := (S*M(D))inverse * S
C     --------------------------------------------------
C
8845  call zgemm('n', 'n', Nh, Nh, mumax, (1.0d0,0.0d0), matS,
```

```
     +           mmm, matM, mmm, (0.0d0,0.0d0), matSM, mmm)
C
C
      call zgeco(matSM, mmm, Nh, ipvt, rcond, z)
      do 90 j=1, mumax
         do 100 i=1, Nh
            z(i)=matS(i,j)
 100     continue
         call zgesl(matSM, mmm, Nh, ipvt, z, 0)
         do 110 i=1, Nh
              matU(i,j)=z(i)
 110     continue
 90   continue
C
      if(progress) write(*, '("Computation of the matrix T")')
C
C     Computation of the matrix T
C     ---------------------------
C
      call zgemm('n', 'n', mumax, mumax, Nh, (1.0d0,0.0d0), matM,
     +           mmm, matU, mmm, (0.0d0,0.0d0), matT, mmm)
C
C     write(9,'(/"The matrix T")')
C     colnr2=0
C200  colnr1=colnr2+1
C     colnr2=colnr2+2
C     if(colnr2.gt.mumax) colnr2=mumax
C     write(9,'(2d18.10, 4x, 2d18.10)') ((matT(i,j), j=colnr1,colnr2),
C    +                                   i=1, mumax)
C     write(9, '(/)')
C     if(colnr2.ne.mumax) goto 200
C
      endif
C
C     Computation of the diagonal matrices W, E and F
C     -----------------------------------------------
C
8850  do 300 i=1, mumax
         dmatW(i)=consta/abs(gammaL(i))
300   continue
C
      if(progress) write(*, '("Computation of the matrix F")')
      call EorF(1,nDstep+1,mumax,Dstep,gammux,mmm,nxm,matF)
C
      if(progress) write(*, '("Computation of the matrix E")')
      call EorF(nDstep+1,nx,mumax,Lstep,gammux,mmm,nxm,matE)
C
C     Computation of the matrix R
C     ---------------------------
C
      do 350 j=1,  mumax
         do 360 i=1, mumax
              matR(i,j)=-matT(i,j)
360      continue
         matR(j,j)=1.0d0-matT(j,j)
350   continue
C
C     Computation of the matrix J
C     ---------------------------
C
8855  call bsnulp(mumax)
C
C     On exit of besnulp
C     zj1(i)       i-th zero of the derivative of the Bessel function J0
C                  (J0'=-J1)
C     zj2(i)       zj1(i)**2
C     matJ(i)      J0(zj1(i))
C
C     Computation of the matrices G en G*Jinvers
C     ------------------------------------------
C
```



```
      if(progress) write(*, '("Computation of the matrix G")')
      do 380 i=1, mumax
         alfa(i)=zj1(i)/a
380   continue
      do 390 j=1, mumax
         do 400 i=1, n+1
            matJ0(i,j)=JB0(alfa(j)*(i-1)*step)
400      continue
390   continue
      do 395 i=1,n+1
          kkwad(i)=kk((i-1)*step,L)**2
395   continue
      do 410 j=1, mumax
         do 412 indx=1, n+1
            hulp1(indx)= matJ0(indx,j)
412      continue
         do 420 i=1, mumax
            do 415 indx=1, n+1
              hulp2(indx)= psiL(indx, i)
415         continue
            matG(i,j)=inprod(n, step, hulp1, hulp2, kkwad)
            GJmin1(i,j)=dcmplx(matG(i,j)/matJ(j))
420      continue
410   continue
C
C     Computation of the matrix D
C     ---------------------------
C
8860  telmax(1)=0
      telmax(2)=0
      telmax(3)=0
      do 440 mu=1,mumax
         do 450 nu=mu, mumax
              matD(mu,nu)=integr()
              matD(nu,mu)=matD(mu,nu)
450      continue
440   continue
C
C     Computation of the matrix K
C     ---------------------------
C     K = (4*k0**2/a)*Minvers*G*Jinnvers*D*(G*Jinvers)'*GAMMA
C     ------------------------------------------------------
C
      if(progress) write(*, '("Computation of the matrix K")')
      alpha=dcmplx(1.0d0)
      beta=dcmplx(0.0d0)
      call zgemm('n', 'n', mumax, mumax, mumax, alpha, GJmin1,
     +           mmm, matD, mmm, beta, mathlp, mmm)
      alpha=dcmplx(4*k0**2/a)
      call zgemm('n', 't', mumax, mumax, mumax, alpha, mathlp,
     +           mmm, GJmin1, mmm, beta, matK, mmm)
      do 460 j=1, mumax
         do 470 i=1,mumax
              matK(i,j)=gammaL(j)*matK(i,j)/dmatW(i)
470      continue
460   continue
C
C     Computation of the matrix R_E (solution of the equation
C     -------------------------------------------------------
C     (K+I)*R_E = K-I )
C     -----------------
C
8865  do 480 j=1,mumax
         do 490 i=1, mumax
              KplusI(i,j)=matK(i,j)
              matRE(i,j)=matK(i,j)
490      continue
         KplusI(j,j)=matK(j,j)+dcmplx(1.0d0)
         matRE(j,j)=matK(j,j)-dcmplx(1.0d0)
480   continue
      call zgeco(KplusI, mmm, mumax, ipvt, rcond, z)
```

```
C     write(*, '("rcond=", d18.10)') rcond
      do 500 j=1,mumax
         do 501 i=1, mumax
              z(i)=matRE(i,j)
501      continue
         call zgesl(KplusI, mmm, mumax, ipvt, z, 0)
         do 505 i=1, mumax
              matRE(i,j)=z(i)
505      continue
500   continue
C
C     Computation of the matrix L (=E * R_E * E) and the matrix I-R*L
C     ---------------------------------------------------------------
8870  do 510 j=1,mumax
         do 520 i=1,mumax
              matL(i,j)=matE(i)*matRE(i,j)*matE(j)
              IminRL(i,j)=dcmplx(0.0)
520      continue
         IminRL(j,j)=dcmplx(1.0)
510   continue
      alpha=dcmplx(-1.0)
      beta=dcmplx(1.0)
      call zgemm('n', 'n',mumax,mumax,mumax, alpha, matR, mmm,
     +           matL, mmm, beta, IminRL, mmm)

C     Computation of the solution c of the equation  (I-R*L)*c = T*a_D
C     ----------------------------------------------------------------
      do 540 i=1,mumax
         vecaD(i)=matF(i)*veca(i)
540   continue
      alpha=dcmplx(1.0)
      beta=dcmplx(0.0)
      call zgemv('n',mumax,mumax,alpha,matT,mmm,vecaD,1,beta,vecc,1)
      call zgeco(IminRL, mmm,mumax, ipvt, rcond, z)
C     print *, 'rcond=', rcond
      call zgesl(IminRL, mmm,mumax, ipvt, vecc, 0)
C
C     Computation of the vector b_D = R*a_D + T*L*c
C     --------------------------------------------
      alpha=dcmplx(1.0)
      beta=dcmplx(0.0)
      call zgemv('n',mumax,mumax,alpha,matR,mmm,vecaD,1,beta,vecbD,1)
      call zgemv('n',mumax,mumax,alpha,matL,mmm,vecc,1,beta,z,1)
      beta=dcmplx(1.0)
      call zgemv('n',mumax,mumax,alpha,matT,mmm,z,1,beta,vecbD,1)
C
C     Computation of the vectors b and b0
C     -----------------------------------
      do 570 i=1,mumax
         vecb(i)=matF(i)*vecbD(i)
         vecb0(i)=matF(i)*matE(i)*veca(i)
570   continue
C
      alpha=dcmplx(1.0)
      beta=dcmplx(0.0)
      call zgemv('n',mumax,mumax,alpha,matRE,mmm,vecb0,1,beta,z,1)
      do 580 i=1,mumax
         vecb0(i)=matF(i)*matE(i)*z(i)
580   continue
C
C     computation of P and P_0   (power and power0)
C     ---------------------------------------------
C
8875  help=2*pi**2*f/(gamma*p0)
      sigma1=0.0d0
      sigma2=0.0d0
      do 590 mu=1, mumax
         if(abs(dimag(gamma0(mu))).lt.1.0d-9) then
              sigma1=sigma1+(abs(veca(mu))**2 -abs(vecb(mu))**2)
         else
              sigma2=sigma2+2*dimag(veca(mu)*conjg(vecb(mu)))
```



```
         endif
590   continue
      power=help*(sigma1+sigma2)
      sigma1=0.0d0
      sigma2=0.0d0
      do 595 mu=1, mumax
         if(abs(dimag(gammaD(mu))).lt.1.0d-9) then
              sigma1=sigma1+(abs(veca(mu))**2 -abs(vecb0(mu))**2)
         else
              sigma2=sigma2+2*dimag(veca(mu)*conjg(vecb0(mu)))
         endif
595   continue
      power0=help*(sigma1+sigma2)
C
C     computation P_in, P_ref and P_tot
C     ---------------------------------
C
8880  do 605 i=1, n+1
         pin(i)=dcmplx(0.0d0)
         pref(i)=dcmplx(0.0d0)
605   continue
      do 600 mu=1, mumax
         do 610 i=1, n+1
              pin(i)=pin(i)+veca(mu)*psi0(i,mu)
              pref(i)=pref(i)+vecb(mu)*psi0(i,mu)
610      continue
600   continue
      do 620 i=1, n+1
         ptot(i)=pin(i)+pref(i)
620   continue
C
C     Printing of output
C     ------------------
C
8885  write(9, '(//"reflection vectors B and B(0)"/
     +          "--------------------------------"//
     +       "mu      B", 36x, "B(0)")')
      do 630 mu=1, maxmu
         write(9,'(i2,2x,2d18.10,2x,2d18.10)') mu,vecb(mu),vecb0(mu)
630   continue
C
      write(9,'(/"Power P=",d18.10,2x,"Power P0=",d18.10,
     +        /"i.e. =  ", d18.10, " dB")')
     +      power,power0, 10.0d0*log10(power0/power)
C
      write(9,'(//"reflection matrix diaphragm"/
     +          "-------------------------"/)')
      colnr2=0
700   colnr1=colnr2+1
      colnr2=colnr2+2
      if(colnr2.gt.maxmu) colnr2=maxmu
      write(9,'(2d18.10,4x,2d18.10)') ((matR(i,j), j=colnr1,colnr2),
     +                                 i=1, maxmu)
      write(9, '(/)')
      if(colnr2.ne.maxmu) goto 700

      write(9,'(//"reflection matrix R_E (end)"/
     +          "-------------------------"/)')
      colnr2=0
710   colnr1=colnr2+1
      colnr2=colnr2+2
      if(colnr2.gt.maxmu) colnr2=maxmu
      write(9,'(2d18.10,4x,2d18.10)') ((matRE(i,j), j=colnr1,colnr2),
     +                                 i=1, maxmu)
      write(9, '(/)')
      if(colnr2.ne.maxmu) goto 710
C
      write(9, '(/"modulus and phase of p_in"/
     +          "-------------------------"/"  r")')
      do 720 i=1,n+1, dnr
         write(9,'(f6.4,d18.10,4x,d18.10)')
```

```
     +        (i-1)*step,abs(pin(i)),fase(pin(i))
720   continue
C
      write(9, '(/"modulus and phase of p_ref"/
     +          "--------------------------"/"  r")')
      do 730 i=1,n+1, dnr
         write(9,'(f6.4,d18.10,4x,d18.10)')
     +        (i-1)*step,abs(pref(i)),fase(pref(i))
730   continue
C
      write(9, '(/"modulus and phase of p_tot"/
     +          "--------------------------"/"  r")')
      do 740 i=1,n+1, dnr
         write(9, '(f6.4,d18.10,4x,d18.10)')
     +        (i-1)*step,abs(ptot(i)),fase(ptot(i))
740   continue
C
      write(9, '(////"distribution of temperature T(x,r)"/
     +          "-----------------------------------")')
      do 760 i=1, nx, dnx
         write(9, '(/"x(",i3,")=",d18.10)') i, x(i)
         colnr2=-dnr
750      colnr1=colnr2+dnr
         colnr2=colnr1+3*dnr
         if(colnr2.gt.n) colnr2=n
         write(9,'(4d18.10)') (funT(x(i),j*step),j=colnr1,colnr2,dnr)
         if(colnr2.ne.n) goto 750
760   continue
9999  write(9,'(//)')
      end
C
C============== End of MAIN  ================================
C
C
C
      double precision function funT(x,r)
      double precision x, r
C
      integer Tchoice
      double precision T0, T1, a, n0, n1, L, nx, term
      common /Tcom/ T0, T1, a, n0, n1, L, Tchoice
C
      term=(1+2.0d0/n1)**(x/L)*(1.0d0+2/n0)**(1.0d0-x/L)
      if(Tchoice.eq.1) then
         nx=2.0d0/(term-1.0d0)
         funT=T1+(T0-T1)*(1.0d0-(r/a)**nx)*term
      else
         funT=T1+(T0-T1)*term
      endif
C
      return
      end
C
C************ End of function funT ****************************
C
C
      double precision function kk(r,x)
      double precision r, x
C
      double precision ckk, k0, Lcopy, funT
      COMMON /kkcom/ ckk, k0, Lcopy
      external funT
C
      if(x.gt.Lcopy) then
         kk=k0
      else
      if(funT(x,r).eq.0.0d0) then
         write(*, '("funT=0", "x=", d18.10, "    r=", d18.10)') x, r
         stop
      endif
         kk=ckk/sqrt(funT(x,r))
```



```
      endif
      return
      end
C
C========= End of function kk(r,x) ================================
C
C
C========= End of file acoustic_ranging.f ==========================
```

```
C
C*********************************************************************
C
      SUBROUTINE BSNULP(MUMAX)
C
      implicit NONE
      INTEGER MMM,MUMAX
      PARAMETER(MMM=30)
C
      REAL*8  PI,KOA,EPSINT,ZJ1(MMM),ZJ2(MMM),JBZ(MMM)
      INTEGER MU,NU,TELMAX(3)
      LOGICAL EERSTE
C
      COMMON//EPSINT,MU,NU,KOA,ZJ1,ZJ2,JBZ,EERSTE,TELMAX,PI
C
      REAL*8 X,F
      INTEGER J
C
      X = 0D0
      ZJ1(1) = 0D0
      ZJ2(1) = 0D0
      JBZ(1)  = 1D0
      DO 10 J = 2, MUMAX
        X = X + 3.141592 D0
        CALL NEWT(X,F)
        ZJ1(J) = X
        ZJ2(J) = X*X
        JBZ(J)  = F
 10   CONTINUE
C
      RETURN
      END
```



```
C
C*********************************************************************
C
      SUBROUTINE CRMBRG(F,A,B,EPS,MAXITER,FLAG,ROMINT)
      implicit NONE
      INTEGER ZAT
      PARAMETER (ZAT=16)
      COMPLEX*16 F,R(0:1,0:ZAT),SM,D,ROMINT
      REAL*8 A,B,H,EPS,ERR
      INTEGER MAXITER,N,M,K,FR
      LOGICAL FLAG
      EXTERNAL F
C
      IF(MAXITER.GT.ZAT) STOP 'CRMBRG'
      N = 0
      M = 1
      H = B-A
      R(1,0) = (F(A)+F(B))*H/2
 10   CONTINUE
      IF (N.LT.MAXITER) THEN
        N = N+1
        H = H/2
        SM = 0
        DO 20 K=0,N-1
          R(0,K) = R(1,K)
 20     CONTINUE
        DO 30 K=1,M
          SM = SM + F(A+(2*K-1)*H)
 30     CONTINUE
        R(1,0) = R(0,0)/2 + H*SM
        M = 2*M
        FR = 1
        DO 40 K=1,N
          FR = FR*4
          D  = R(1,K-1)-R(0,K-1)
          R(1,K) = R(1,K-1) + D/(FR-1)
          IF(R(1,K).NE.0D0) THEN
            ERR = CDABS(D/R(1,K))/FR
          ELSE
            ERR = CDABS(D)/FR
          ENDIF
          IF ((ERR.LT.EPS).AND.(N.GT.3)) THEN
            ROMINT = R(1,K)
            FLAG = .TRUE.
            RETURN
          ENDIF
 40     CONTINUE
      ELSE
        ROMINT = R(1,N)
        FLAG = .FALSE.
        RETURN
      ENDIF
      GOTO 10
      END
```

```
      subroutine EIGAM (x, a, n,  ierr, mumax, gamma2)
      implicit NONE
      integer n, nm, ierr, mumax, mmm
      parameter(nm=1001, mmm=100)
      double precision x, a
      double precision gamma2(mumax), psi(nm,mmm)
C
C     Bepaling van de kwadraten van de eigenwaarden gamma(m).
C     22 april 1992
C
C     nm    : rij-dimensie van twee-dimensionale array parameters
C             zoals gedeclareerd in het hoofdprogramma
C     mumax : aantal te bepalen eigenwaarden
C     mmm   : bovengrens voor mumax
C     ierr  : foutindikator is bij succesvolle afloop gelijk aan 0
C
      integer   i, m, aantal
      double precision k(nm), alfa(nm),halfa(nm),e(nm),d(nm),e2(nm),
     +       rv1(nm),rv2(nm),rv3(nm),rv4(nm),rv5(nm),rv6(nm),
     +       p(-1:nm), kk
C
      external gsturm, kk
C     halfa(i) := alfa(i-0.5d0)
C
      double precision knul,lb,ub,og,bg,mid,half,h,h2, eps1
C
      half=0.5d0
      knul=kk(0.0d0,x)
      h=a/n
      do 10 i=1, n+1
         k(i)=kk(i*h,x)
         alfa(i)=sqrt(i*h)/k(i)
         halfa(i)=sqrt((i-half)*h)/kk((i-half)*h,x)
 10   continue
      e(2)=-2.0d0*halfa(1)/alfa(1)
      e2(2)=e(2)**2
      do 20 i=3, n
         e(i)=-halfa(i-1)**2/(alfa(i-1)*alfa(i-2))
         e2(i)=e(i)**2
 20   continue
      e(n+1)=-halfa(n)*sqrt(halfa(n)**2+halfa(n+1)**2)/
     +       (alfa(n)*alfa(n-1))
      e2(n+1)=e(n+1)**2
      h2=h**2
      d(1)=4-h2*knul**2
      do 30 i=2, n+1
         d(i)=(halfa(i-1)**2+halfa(i)**2)/alfa(i-1)**2-h2*k(i-1)**2
 30   continue
C
C
C     berekening van ub
C
      bg=abs(d(1))+abs(e(2))
      do 40 i=2, n
         bg=max(bg, abs(d(i))+abs(e(i))+abs(e(i+1)))
 40   continue
      bg=max(bg, abs(d(n+1))+abs(e(n+1)))
C
C     Alle eigenwaarden liggen in het interval [-bg,bg]
C
      lb=-bg
      og=-bg
      p(-1)=0d0
      p(0)=1d0
 90   mid=(og+bg)/2d0
      aantal=0
      do 80 i=1,n+1
         p(i)=(d(i)-mid)*p(i-1)-e2(i)*p(i-2)
         if(sign(1,p(i)).ne.sign(1,p(i-1))) aantal=aantal+1
 80   continue
      if(aantal.eq.mumax) then
```



```
         ub=mid
      else
         if(aantal.lt.mumax) then
            og=mid
         else
            bg=mid
         endif
         goto 90
      endif
C
      eps1=-1.0d0
C
      call gsturm(nm, n+1,eps1,d, e,e2,lb,ub,mumax, m, gamma2, psi,
     +            ierr, rv1, rv2, rv3, rv4, rv5, rv6)
      if(ierr.ne.0.or.m.ne.mumax) then
         write(*, '("ierr=", i5,/ "m=", i10)') ierr, m
         return
      endif
C
C
      do 100 i=1, mumax
         gamma2(i)=gamma2(i)/h2
 100  continue
C
      return
      end
```

```
      subroutine EIGAMM (em, x, a, n,  ierr, mumax, gamma2)
      implicit NONE
      integer n, nm, ierr, mumax, em, mmm
      parameter(nm=1001, mmm=100)
      double precision x, a
      double precision gamma2(mumax), psi(nm,mmm)
C
C     Bepaling van de kwadraten van de eigenwaarden gamma(m).
C     22 april 1992
C
C     nm    : rij-dimensie van twee-dimensionale array parameters
C             zoals gedeclareerd in het hoofdprogramma
C     mmm   : bovengrens voor mumax
C     ierr  : foutindikator is bij succesvolle afloop gelijk aan 0
C
      integer   i, m, aantal
      double precision k(nm), alfa(nm),halfa(nm),e(nm),d(nm),e2(nm),
     +       rv1(nm),rv2(nm),rv3(nm),rv4(nm),rv5(nm),rv6(nm),
     +       p(-1:nm), kk
C
      external gsturm, kk
C     halfa(i) := alfa(i-0.5d0)
C
      double precision lb,ub,og,bg,mid,half,h,h2, eps1
C
      half=0.5d0
      h=a/n
      do 10 i=1, n+1
         k(i)=kk(i*h,x)
         alfa(i)=sqrt(i*h)/k(i)
         halfa(i)=sqrt((i-half)*h)/kk((i-half)*h,x)
 10   continue
      do 20 i=2, n-1
         e(i)=-halfa(i)**2/(alfa(i)*alfa(i-1))
         e2(i)=e(i)**2
 20   continue
      e(n)=-halfa(n)*sqrt(halfa(n)**2+halfa(n+1)**2)/
     +       (alfa(n)*alfa(n-1))
      e2(n)=e(n)**2
      h2=h**2
      do 30 i=1, n
         d(i)=(halfa(i)**2+halfa(i+1)**2)/alfa(i)**2-h2*k(i)**2
     +        +(em/real(i))**2
 30   continue
C
C
C     berekening van ub
C
      bg=abs(d(1))+abs(e(2))
      do 40 i=2, n-1
         bg=max(bg, abs(d(i))+abs(e(i))+abs(e(i+1)))
 40   continue
      bg=max(bg, abs(d(n))+abs(e(n)))
C
C     Alle eigenwaarden liggen in het interval [-bg,bg]
C
      lb=-bg
      og=-bg
      p(-1)=0d0
      p(0)=1d0
 90   mid=(og+bg)/2d0
      aantal=0
      do 80 i=1,n
         p(i)=(d(i)-mid)*p(i-1)-e2(i)*p(i-2)
         if(sign(1,p(i)).ne.sign(1,p(i-1))) aantal=aantal+1
 80   continue
      if(aantal.eq.mumax) then
         ub=mid
      else
         if(aantal.lt.mumax) then
              og=mid
```



```
         else
              bg=mid
         endif
         goto 90
      endif
C
      eps1=-1.0d0
C
      call gsturm(nm, n, eps1, d, e,e2,lb,ub,mumax, m, gamma2, psi,
     +           ierr, rv1, rv2, rv3, rv4, rv5, rv6)
      if(ierr.ne.0.or.m.ne.mumax) then
         write(*, '("ierr=", i5,/ "m=", i10)') ierr, m
         return
      endif
C
      do 100 i=1, mumax
         gamma2(i)=gamma2(i)/h2
 100  continue
C
      return
      end
```

```
      subroutine EIGEN (x, a, n,consta,ierr,mumax, gamma2, psi)
C
C     Computation of the modes and squares of the eigenvalues.
C
      implicit NONE
      integer n, nm, ierr, mumax
      parameter(nm=1001)
      double precision x, a, consta
      double precision gamma2(mumax), psi(nm,mumax)
C
C
      integer   i, j, m, aantal
      double precision k(nm), alfa(nm),halfa(nm),e(nm),d(nm),e2(nm),
     +       rv1(nm),rv2(nm),rv3(nm),rv4(nm),rv5(nm),rv6(nm),
     +       p(-1:nm), gamma(nm), kk, f
C
      external tsturm, kk
C     halfa(i) := alfa(i-0.5)
C
      double precision knul,lb,ub,og,bg,mid,s,half,h,h2, eps1
C
      half=0.5d0
      knul=kk(0.0d0,x)
      h=a/n
      do 10 i=1, n+1
         k(i)=kk(i*h,x)
         alfa(i)=sqrt(i*h)/k(i)
         halfa(i)=sqrt((i-half)*h)/kk((i-half)*h,x)
 10   continue
      e(2)=-2.0d0*halfa(1)/alfa(1)
      e2(2)=e(2)**2
      do 20 i=3, n
         e(i)=-halfa(i-1)**2/(alfa(i-1)*alfa(i-2))
         e2(i)=e(i)**2
 20   continue
      e(n+1)=-halfa(n)*sqrt(halfa(n)**2+halfa(n+1)**2)/
     +       (alfa(n)*alfa(n-1))
      e2(n+1)=e(n+1)**2
      h2=h**2
      d(1)=4-h2*knul**2
      do 30 i=2, n+1
         d(i)=(halfa(i-1)**2+halfa(i)**2)/alfa(i-1)**2-h2*k(i-1)**2
 30   continue
C
C
C     berekening van ub
C
      bg=abs(d(1))+abs(e(2))
      do 40 i=2, n
         bg=max(bg, abs(d(i))+abs(e(i))+abs(e(i+1)))
 40   continue
      bg=max(bg, abs(d(n+1))+abs(e(n+1)))
C
C     Alle eigenwaarden liggen in het interval [-bg,bg]
C
      lb=-bg
      og=-bg
      p(-1)=0d0
      p(0)=1d0
 90   mid=(og+bg)/2d0
      aantal=0
      do 80 i=1,n+1
         p(i)=(d(i)-mid)*p(i-1)-e2(i)*p(i-2)
         if(sign(1,p(i)).ne.sign(1,p(i-1))) aantal=aantal+1
 80   continue
      if(aantal.eq.mumax) then
         ub=mid
      else
         if(aantal.lt.mumax) then
              og=mid
         else
```



```
              bg=mid
         endif
         goto 90
      endif
C
      eps1=-1.0d0
C
      call tsturm(nm, n+1,eps1, d,e,e2,lb,ub,mumax, m, gamma2, psi,
     +            ierr, rv1, rv2, rv3, rv4, rv5, rv6)
      if(ierr.ne.0.or.m.ne.mumax) then
         write(*, '("ierr=", i5,/ "m=", i10)') ierr, m
         return
      endif
C
C Transformatie van psi naar eigenvektor van de niet-symmetrische
C discretisatie-matrix.
C
      do 110 i=1,n+1
         if(i.eq.1) then
            s=halfa(1)/2.0d0
         else
            if(i.eq.n+1) then
               s=halfa(n)*alfa(n)/sqrt(halfa(n)**2+halfa(n+1)**2)
            else
               s=alfa(i-1)
            endif
         endif
         do 120, j=1, mumax
            psi(i,j)=psi(i,j)/s
 120     continue
 110  continue
C
      do 100 i=1, mumax
         gamma2(i)=gamma2(i)/h2
         gamma(i)=sqrt(abs(gamma2(i)))
 100  continue
C
C     Schaling van psi
C
      do 50 j=1, mumax
         s=n*(psi(n+1,j)/k(n))**2
         f=4.0d0
         do 60 i=2, n
            s=s+f*(i-1)*(psi(i,j)/k(i-1))**2
            if(f.eq.4.0d0) then
              f=2.0d0
            else
              f=4.0d0
            endif
 60      continue
         s=gamma(j)*h2*s/3
         s=sqrt(consta/s)
         if(psi(1,j).lt.0) s=-s
         do 70 i=1,n+1
            psi(i,j)=s*psi(i,j)
 70      continue
 50   continue
      return
      end
```

```
      subroutine EorF(lbj,ubj,mumax,step,gammux,mmm,nxm,matE)
      implicit NONE
      integer lbj, ubj, mumax, mmm, nxm
      double precision step
      double complex gammux(mmm,nxm), matE(mumax)
C
C     computation of the diagonal of the matrix E or F.
C
C     lbj and ubj are the lower bound and upperbound of the index j
C     that points to the j-th integration subinterval.
C
C     step is integration step.
C
      implicit NONE
      integer i , j
      double precision f
      double complex iu, help
C
      iu=dcmplx(0.0d0,1.0d0)
      do 300 i=1, mumax
         matE(i)=gammux(i,lbj)
300   continue
      do 310 i=1, mumax
         matE(i)=matE(i)+gammux(i, ubj)
310   continue
      f=4.0d0
      do 320 j=lbj+1, ubj-1
         do 330 i=1, mumax
              matE(i)=matE(i)+f*gammux(i, j)
330      continue
         if(f.eq.4.0d0) then
                f=2.0d0
         else
                f=4.0d0
         endif
320   continue
      do 360 i=1,mumax
            help=-iu*step*matE(i)/3
            if(dble(help).gt.-228d0) then
                matE(i)=exp(help)
            else
                matE(i)=0.0d0
            endif
360   continue
      return
      end
```



```
      double precision function epsiln (x,machep)
      implicit NONE
      double precision x, machep
      epsiln = machep*dabs(x)
      return
      end
```

```
      double precision function epslon (x,machep)
      implicit NONE
      double precision x, machep
      epslon = machep*dabs(x)
      return
      end
```



```
      double precision function fase(p)
      implicit NONE
      double complex p,rad
C
      rad = 57.29577 95130 82320 87679 81548 14105 17033 24054 72 D0
      if(p.ne.dcmplx(0.0d0)) then
         fase = atan2(dimag(p),dble(p))*rad
      else
         fase = 0.0d0
      endif
      return
      end
```

```
      subroutine GAMSLM(em, x, hs, n, ierr, mm, gamma)
C
C     Computation of the eigenvalues for m > 0
C     em is the parameter m in the differential equation
C     20 march 1992
C
      implicit NONE
      integer em, n,ierr,mm,j, mmm
      parameter(mmm=100)
C
C mmm is een bovengrens voor mm
C n * h = hs
C n dient even te zijn
C
      double precision x, gamma1(mmm),gamma2(mmm), hs
      double complex gamma(mm)
C
      external EIGAMM
C
      call EIGAMM(em, x, hs, n, ierr, mm, gamma1)
      call EIGAMM(em, x, hs, 2*n,  ierr, mm, gamma2)
C
C
C   h**2-extraplolatie
C
      do 15 j=1,mm
         gamma(j)=dcmplx(0.0d0,-1.0d0)*
     +            sqrt(dcmplx(gamma2(j)+(gamma2(j)-gamma1(j))/3.0d0))
 15   continue
C
      end
```



```
      subroutine GAMSOL(x, h, n, ierr, mumax, gamma)
C
C     Computation of the eigenvalue for m = 0
C     By m is meant the parameter m in the differential equation.
C     22 april 1992
C
      implicit NONE
      integer n,ierr,mumax,j, mmm
      parameter(mmm=100)
C
C mmm is een bovengrens voor mumax
C n dient even te zijn
C
      double precision x, gamma1(mmm),gamma2(mmm), h
      double complex gamma(mumax)
C
      external EIGAM
C
      call EIGAM(x, h, n, ierr, mumax, gamma1)
      call EIGAM(x, h, 2*n,  ierr, mumax, gamma2)
C
C
C
      do 15 j=1,mumax
         gamma(j)=dcmplx(0.0d0,-1.0d0)*
     +            sqrt(dcmplx(gamma2(j)+(gamma2(j)-gamma1(j))/3.0d0))
 15   continue
C
      end
```

```
      subroutine gsturm(nm,n,eps1,d,e,e2,lb,ub,mm,m,w,z,
     x                  ierr,rv1,rv2,rv3,rv4,rv5,rv6)
c
      implicit NONE
      integer i,k,m,n,p,q,r,s,ii,mm,m1,m2,nm,
     x        ierr,isturm
      double precision d(n),e(n),e2(n),w(mm),z(nm,mm),
     x       rv1(n),rv2(n),rv3(n),rv4(n),rv5(n),rv6(n)
      double precision u,v,lb,t1,t2,ub,xu,x0,x1,eps1,
     x       tst1,tst2,epsiln, machep
c
c     Deze routine berekent in tegenstelling tot de routine tsturm
C     slechts de eigenwaarden en niet de bijbehorende eigenvektoren.
C
c
c     ------------------------------------------------------------------
c
      machep=1.0d0
 1234 machep=machep/2.0d0
      if(machep+1.0d0.ne.1.0d0) goto 1234
      machep=machep*2.0d0
c
      ierr = 0
      t1 = lb
      t2 = ub
c     ......... look for small sub-diagonal entries .........
      do 40 i = 1, n
         if (i .eq. 1) go to 20
         tst1 = dabs(d(i)) + dabs(d(i-1))
         tst2 = tst1 + dabs(e(i))
         if (tst2 .gt. tst1) go to 40
   20    e2(i) = 0.0d0
   40 continue
c     ......... determine the number of eigenvalues
c               in the interval .........
      p = 1
      q = n
      x1 = ub
      isturm = 1
      go to 320
   60 m = s
      x1 = lb
      isturm = 2
      go to 320
   80 m = m - s
      if (m .gt. mm) go to 980
      q = 0
      r = 0
c     ......... establish and process next submatrix, refining
c               interval by the gerschgorin bounds .........
  100 if (r .eq. m) go to 1001
      p = q + 1
      xu = d(p)
      x0 = d(p)
      u = 0.0d0
c
      do 120 q = p, n
         x1 = u
         u = 0.0d0
         v = 0.0d0
         if (q .eq. n) go to 110
         u = dabs(e(q+1))
         v = e2(q+1)
  110    xu = dmin1(d(q)-(x1+u),xu)
         x0 = dmax1(d(q)+(x1+u),x0)
         if (v .eq. 0.0d0) go to 140
  120 continue
c
  140 x1 = epsiln(dmax1(dabs(xu),dabs(x0)),machep)
      if (eps1 .le. 0.0d0) eps1 = -x1
      if (p .ne. q) go to 180
```



```
c     ......... check for isolated root within interval .........
      if (t1 .gt. d(p) .or. d(p) .ge. t2) go to 940
      r = r + 1
c
      do 160 i = 1, n
  160 z(i,r) = 0.0d0
c
      w(r) = d(p)
      z(p,r) = 1.0d0
      go to 940
  180 u = q-p+1
      x1 = u * x1
      lb = dmax1(t1,xu-x1)
      ub = dmin1(t2,x0+x1)
      x1 = lb
      isturm = 3
      go to 320
  200 m1 = s + 1
      x1 = ub
      isturm = 4
      go to 320
  220 m2 = s
      if (m1 .gt. m2) go to 940
c     ......... find roots by bisection .........
      x0 = ub
      isturm = 5
c
      do 240 i = m1, m2
         rv5(i) = ub
         rv4(i) = lb
  240 continue
c     ......... loop for k-th eigenvalue
c               for k=m2 step -1 until m1 do --
c               (-do- not used to legalize -computed go to-) .........
      k = m2
  250    xu = lb
c     ......... for i=k step -1 until m1 do -- .........
         do 260 ii = m1, k
            i = m1 + k - ii
            if (xu .ge. rv4(i)) go to 260
            xu = rv4(i)
            go to 280
  260    continue
c
  280    if (x0 .gt. rv5(k)) x0 = rv5(k)
c     ......... next bisection step .........
  300    x1 = (xu + x0) * 0.5d0
         if ((x0 - xu) .le. dabs(eps1)) go to 420
         tst1 = 2.0d0 * (dabs(xu) + dabs(x0))
         tst2 = tst1 + (x0 - xu)
         if (tst2 .eq. tst1) go to 420
c     ......... in-line procedure for sturm sequence .........
  320    s = p - 1
         u = 1.0d0
c
         do 340 i = p, q
            if (u .ne. 0.0d0) go to 325
            v = dabs(e(i)) / epsiln(1.0d0, machep)
            if (e2(i) .eq. 0.0d0) v = 0.0d0
            go to 330
  325       v = e2(i) / u
  330       u = d(i) - x1 - v
            if (u .lt. 0.0d0) s = s + 1
  340    continue
c
         go to (60,80,200,220,360), isturm
c     ......... refine intervals .........
  360    if (s .ge. k) go to 400
         xu = x1
         if (s .ge. m1) go to 380
         rv4(m1) = x1
```

```
         go to 300
  380    rv4(s+1) = x1
         if (rv5(s) .gt. x1) rv5(s) = x1
         go to 300
  400    x0 = x1
         go to 300
c     ......... k-th eigenvalue found ..........
  420    rv5(k) = x1
         w(k-m1+1)=x1
       k = k - 1
       if (k .ge. m1) goto 250
  940  if(q.lt.n) goto 100
       goto 1001
980    ierr=3*n+r
 1001  lb = t1
       ub = t2
       return
       end
```



```
c
c ***********************************************************************
c
      SUBROUTINE H1B(ZZ,NN,H1N,DH1,H11,FLAG)
c     in: ZZ  : complex argument
c         NN  : order of bessel functions
c     out: H1N,DH1,H11: Hankel function values H1n, dH1n, H1n+1
c
c     this subroutine calculates Hankel functions H1
c     of order N and with complex argument
c
      implicit NONE
      COMPLEX*16 Z,ZZ,H1N,DH1,H11,I,CN,KN,DK,K1,JN,DJ,J1
      REAL*8 PI
      INTEGER N,NN
      LOGICAL FLAG
      DATA PI /3.141 592 653 589 793 238 462 643 383 279 D0/
      Z = ZZ
      N = NN
      FLAG =.FALSE.
c
c     flag is used to indicate true or false returned value.
c     return if argument is 0.
c
      IF(Z.EQ.(0D0,0D0)) RETURN
c
c     the H1 function values are calculated by means of the following
c     formulas:
c     h1(z,n)=          2/pi*(-i)**(n+1)*k(-i*z,n)     ( 0  < arg z <= pi )
c     h1(z,n)= 2*j(z,n)-2/pi*  i **(n+1)*k( i*z,n)     ( -pi< arg z <= 0  )
c     if (im(z)>0) or (im(z)=0 and re(z)<0) then
c     the first formula is used, otherwise the last
c     (not inefficient because jn is not calculated in kb)
c
      I = (0D0,1D0)
      CN = 2D0/PI
      IF ((DIMAG(Z).GT.0D0).OR.((DIMAG(Z).EQ.0D0).AND.
     >                                  (DBLE(Z).LT.0D0))) THEN
        CALL KB(-I*Z,N,KN,DK,K1,FLAG)
        IF(.NOT.FLAG) RETURN
        GOTO (10,11,12,20) MOD(N,4)+1
 10       CN = DCMPLX(0D0,-CN)
          GOTO 20
 11       CN = -CN
          GOTO 20
 12       CN = DCMPLX(0D0,CN)
 20       CONTINUE
        H1N = CN*KN
        DH1 = -I*CN*DK
        H11 = -I*CN*K1
      ELSE
        CALL KB(I*Z,N,KN,DK,K1,FLAG)
        IF(.NOT.FLAG) RETURN
        CALL JB(Z,N,JN,DJ,J1,FLAG)
        IF(.NOT.FLAG)RETURN
        GOTO (30,31,32,40) MOD(N,4)+1
 30       CN = DCMPLX(0D0,CN)
          GOTO 40
 31       CN = -CN
          GOTO 40
 32       CN = DCMPLX(0D0,-CN)
 40       CONTINUE
        H1N = 2D0*JN - CN*KN
        DH1 = 2D0*DJ - I*CN*DK
        H11 = 2D0*J1 - I*CN*K1
      ENDIF
      FLAG = .TRUE.
      RETURN
      END
```

```
c
c ***********************************************************************
c
      SUBROUTINE H2B(ZZ,NN,H2N,DH2,H21,FLAG)
c     in: ZZ  : complex argument
c         NN  : order of bessel functions
c     out: H2N,DH2,H21: Hankel function values H2n, dH2n, H2n+1
c
c     this subroutine calculates Hankel functions H2
c     of order N and with complex argument
c
      implicit NONE
      COMPLEX*16 Z,ZZ,H2N,DH2,H21,I,CN,KN,DK,K1,JN,DJ,J1
      REAL*8 PI
      INTEGER N,NN
      LOGICAL FLAG
      DATA PI /3.141 592 653 589 793 238 462 643 383 279 D0/
      Z = ZZ
      N = NN
      FLAG =.FALSE.
c
c     flag is used to indicate true or false returned value.
c     return if argument is 0.
c
      IF(Z.EQ.(0D0,0D0)) RETURN
c
c     the H2 function values are calculated by means of the following
c     formulas:
c     h2(z,n)= 2*j(z,n)-2/pi*(-i)**(n+1)*k(-i*z,n)     ( 0  < arg z <= pi )
c     h2(z,n)=          2/pi*  i **(n+1)*k( i*z,n)     ( -pi< arg z <= 0  )
c     if (im(z)>0) or (im(z)=0 and re(z)<0) then
c     the first formula is used, otherwise the last
c     (not inefficient because jn is not calculated in kb)
c
      I = (0D0,1D0)
      CN = 2D0/PI
      IF ((DIMAG(Z).GT.0D0).OR.((DIMAG(Z).EQ.0D0).AND.
     >                          (DBLE(Z).LT.0D0))) THEN
        CALL KB(-I*Z,N,KN,DK,K1,FLAG)
        IF(.NOT.FLAG) RETURN
        CALL JB(Z,N,JN,DJ,J1,FLAG)
        IF(.NOT.FLAG) RETURN
        GOTO (10,11,12,20) MOD(N,4)+1
 10       CN = DCMPLX(0D0,-CN)
          GOTO 20
 11       CN = -CN
          GOTO 20
 12       CN = DCMPLX(0D0,CN)
 20       CONTINUE
        H2N = 2D0*JN - CN*KN
        DH2 = 2D0*DJ + I*CN*DK
        H21 = 2D0*J1 + I*CN*K1
      ELSE
        CALL KB(I*Z,N,KN,DK,K1,FLAG)
        IF(.NOT.FLAG) RETURN
        GOTO (30,31,32,40) MOD(N,4)+1
 30       CN = DCMPLX(0D0,CN)
          GOTO 40
 31       CN = -CN
          GOTO 40
 32       CN = DCMPLX(0D0,-CN)
 40       CONTINUE
        H2N = CN*KN
        DH2 = I*CN*DK
        H21 = I*CN*K1
      ENDIF
      FLAG = .TRUE.
      RETURN
      END
```



```
c
c ***********************************************************************
c
      SUBROUTINE IB(ZZ,NN,IN,DI,I1,FLAG)
c
c     in: ZZ,NN
c     out: IN=In,DI=dIn/dz,IN1=In+1,FLAG
c
c     this subroutine calculates modified bessel functions
c     of the first kind of order N and complex argument Z
c     this is done by using the function Jn = ordinary
c     bessel function of the first kind
c     the following relation is used:
c     in(z)=(-i)**n * jn(i*z)
c
      implicit NONE
      COMPLEX*16 ZZ,Z,IN,DI,I1,CM,CM1,JN,DJ,J1
      INTEGER N,NN
      LOGICAL FLAG
      Z = DCMPLX(-DIMAG(ZZ),DBLE(ZZ))
c     z = i*zz
      N = IABS(NN)
      CALL JB(Z,N,JN,DJ,J1,FLAG)
c
c     flag indicates true or false returned JB-values
c
      IF(.NOT.FLAG) RETURN
c
c     calculate cm=(-i)**n, cm1=(-i)**(n+1)
c
      GOTO (10,11,12,13) MOD(N,4)+1
 10    CM  = ( 1.0D0, 0.0D0)
       CM1 = ( 0.0D0,-1.0D0)
       GOTO 20
 11    CM  = ( 0.0D0,-1.0D0)
       CM1 = (-1.0D0, 0.0D0)
       GOTO 20
 12    CM  = (-1.0D0, 0.0D0)
       CM1 = ( 0.0D0, 1.0D0)
       GOTO 20
 13    CM  = ( 0.0D0, 1.0D0)
       CM1 = ( 1.0D0, 0.0D0)
 20    CONTINUE
      IN =  CM *JN
      DI = -CM1*DJ
      I1 =  CM1*J1
      RETURN
      END
```

```
      double precision function inprod(nhs,step,psi1,psi2,kkwad)
      implicit NONE
      integer nhs, l
      double precision step,psi1(nhs+1),psi2(nhs+1),kkwad(nhs+1),s,f

      s=nhs*psi1(nhs+1)*psi2(nhs+1)/kkwad(nhs+1)
      f=4.0d0
      do 10 l=2,nhs
         s=s+f*(l-1)*psi1(l)*psi2(l)/kkwad(l)
         if(f.eq.4.0d0) then
              f=2.0d0
         else
              f=4.0d0
         endif
10    continue
      inprod=step**2*s/3.0d0
      return
      end
```



```
C
C*****************************************************************************
C
      COMPLEX*16 FUNCTION INTEGR()
C
      implicit NONE
      INTEGER MMM
      PARAMETER(MMM=30)
C
      REAL*8  PI,K0A,EPSINT,ZJ1(MMM),ZJ2(MMM),JBZ(MMM)
      INTEGER MU,NU,TELMAX(3)
      LOGICAL EERSTE
C
      COMMON//EPSINT,MU,NU,K0A,ZJ1,ZJ2,JBZ,EERSTE,TELMAX,PI
C
      REAL*8 TGRND1,TGRND2,I1,I2
      COMPLEX*16 TGRND3,I3,TOTAL,CU
      REAL*8 A,B
      INTEGER MAXITER
      LOGICAL FLAG
      EXTERNAL TGRND1,TGRND2,TGRND3
C
      CU = DCMPLX(0D0,1D0)
      A = 0D0
      B = K0A
      MAXITER = 10
      EERSTE = .TRUE.
      CALL ROMBRG(TGRND1,A,B,EPSINT,MAXITER,FLAG,I1)
C     WRITE(*,'(3I5,F15.10)')MU,NU,TELMAX(1),I1
      IF (.NOT.FLAG) THEN
        WRITE(*,*)' GEEN CONVERGENTIE IN INTEGRAAL 1'
        WRITE(*,*)
      ENDIF
      B = 1/K0A
      EERSTE = .TRUE.
      CALL ROMBRG(TGRND2,A,B,EPSINT,MAXITER,FLAG,I2)
C     WRITE(*,'(3I5,F15.10)')MU,NU,TELMAX(2),I2
      IF (.NOT.FLAG) THEN
        WRITE(*,*)' GEEN CONVERGENTIE IN INTEGRAAL 2'
      ENDIF
      B = PI/2D0
      EERSTE = .TRUE.
      MAXITER = 10
      CALL CRMBRG(TGRND3,A,B,EPSINT,MAXITER,FLAG,I3)
C     WRITE(*,'(3I5,2F15.10)')MU,NU,TELMAX(3),I3
      IF (.NOT.FLAG) THEN
        WRITE(*,*)' GEEN CONVERGENTIE IN INTEGRAAL 3'
      ENDIF
      TOTAL = -CU*2D0/PI * (I1 + I2) + I3
      IF (MU.EQ.NU) THEN
        IF(MU.EQ.1) THEN
          TOTAL = TOTAL + CU * DLOG(1D0+DSQRT(2D0))/PI/K0A
        ELSE
          TOTAL = TOTAL + CU * 0.5D0/DSQRT(ZJ2(MU)-K0A*K0A)
        ENDIF
      ENDIF
      INTEGR = TOTAL
C
      RETURN
      END
```

```
c
c ***********************************************************************
c
      SUBROUTINE JB(ZZ,NN,JN,DJ,J1,FLAG)
c
c               ZZ = complex argument
c               NN = order
c          JN,DJ,J1 = Bessel functions Jn, dJn, Jn+1
c              FLAG = logical indicating true or false returned JN-value
c
      implicit NONE
      INTEGER EMAX,EMIN
      PARAMETER(EMAX = 308,EMIN = -278)
      COMPLEX*16 Z,ZZ,JN,DJ,J1
      INTEGER N,NN,LM,M,M2
      REAL*8 SUMR,SUMI,T0R,T0I,T1R,T1I,T2R,T2I,DSMIN,DSMAX,
     *       JNR,JNI,J1R,J1I,ZR,ZI,ZAR,ZAI,ZAB,ZMIN,ZMAX
      LOGICAL SIGN,FLAG,NNEG,NODD,RZNEG,CONJ
c
c     this subroutine calculates complex Bessel functions of the first kind
c     of order n
c     the algorithm is based on the recurrence relation
c      J(z,n-1) = (2*n/z)*J(z,n)-J(z,n+1)
c
c     nearly all calculations are split up in real and imaginary part
c
      N = NN
      Z = ZZ
c
c     return JB directly if Z = 0+0i
c
      JN = (0.0D0,0.0D0)
      DJ = (0.0D0,0.0D0)
      J1 = (0.0D0,0.0D0)
      IF (Z.EQ.(0.0D0,0.0D0)) THEN
         FLAG = .TRUE.
         IF (N.EQ.0)  JN = ( 1.0D0,0.0D0)
         IF (N.EQ.1)  DJ = ( 0.5D0,0.0D0)
         IF (N.EQ.-1) DJ = (-0.5D0,0.0D0)
         RETURN
      ENDIF
c
c     initialize the logical variable flag
c     FLAG will be set to .TRUE. if JN can be calculated
c
      FLAG = .FALSE.
c
c     calculate logical constants
c
      IF (N.LT.0) THEN
                     NNEG = .TRUE.
                        N = - N
      ELSE
                     NNEG = .FALSE.
      ENDIF
      IF(MOD(N,2).NE.0) THEN
                          NODD = .TRUE.
      ELSE
                          NODD = .FALSE.
      ENDIF
      IF (DBLE(Z).LT.0.0D0) THEN
                           RZNEG = .TRUE.
                               Z = - Z
      ELSE
                           RZNEG = .FALSE.
      ENDIF
      IF (NODD.AND.(NNEG.NEQV.RZNEG)) THEN
                            SIGN = .TRUE.
      ELSE
                            SIGN = .FALSE.
      ENDIF
```



```
      IF (DIMAG(Z).LT.0.0D0) THEN
                                CONJ = .TRUE.
                                   Z = DCONJG(Z)
      ELSE
                                CONJ = .FALSE.
      ENDIF
c
c     calculate constants
c
      ZR = DBLE(Z)
      ZI = DIMAG(Z)
      ZAB = ZR*ZR + ZI*ZI
      ZAR = ZR/ZAB
      ZAI = ZI/ZAB
c
c  calculate LM
c
      LM = INT(1. + 3.*ZAB**(1./24.) + 9.*ZAB**(1./6.) +
     *             DMAX1(DBLE(N),DSQRT(ZAB)))
      IF (LM.GT.299) RETURN
c
c  calculate DSMIN (for avoiding underflow)
c
c  calculate DSMAX (for avoiding overflow)
c
      ZMAX = DMAX1(DABS(ZAR),DABS(ZAI))
      ZMIN = DMIN1(DABS(ZAR),DABS(ZAI))
      IF (ZMIN.EQ.0.0D0) ZMIN = ZMAX
      DSMIN = 10.0D0**MAX(EMIN,EMIN-NINT(DLOG10(ZMIN)-1D-8))
      DSMAX = 10.0D0**MIN(EMAX,EMAX-NINT(DLOG10(2.*LM*ZMAX)-1D-8))
c
c   function values by means of the recurrence relation.
c
      SUMR = 0.0D0
      SUMI = 0.0D0
      T1R  = 0.0D0
      T1I  = 0.0D0
      IF (ZR.EQ.0.0D0) THEN
                          T0R = 0.0D0
      ELSE
                          T0R = DSMIN
      ENDIF
      IF (ZI.EQ.0.0D0) THEN
                          T0I = 0.0D0
      ELSE
                          T0I = DSMIN
      ENDIF
      DO 30 M = LM,0,-1
         T2R = T0R
         T2I = T0I
         M2  = 2*(M+1)
         T0R = M2*(ZAR*T2R+ZAI*T2I)-T1R
         T0I = M2*(ZAR*T2I-ZAI*T2R)-T1I
         T1R = T2R
         T1I = T2I
c
c     return if there will be overflow
c
         IF (DABS(T0R).GT.DSMAX .OR. DABS(T0I).GT.DSMAX) RETURN
         IF (M.EQ.N) THEN
                           JNR = T0R
                           JNI = T0I
                           J1R = T1R
                           J1I = T1I
         ENDIF
         GOTO (21,22,23,24) 1 + MOD(M,4)
   21    IF (M.EQ.0) THEN
                           SUMR = 2*SUMR + T0R
                           SUMI = 2*SUMI + T0I
         ELSE
                           SUMR = SUMR + T0R
```

```
                        SUMI = SUMI + TOI
        ENDIF
        GOTO 30
   22   IF (M.EQ.0) THEN
                        SUMR = 2*SUMR + TOI
                        SUMI = 2*SUMI - TOR
        ELSE
                        SUMR = SUMR + TOI
                        SUMI = SUMI - TOR
        ENDIF
        GOTO 30
   23   IF (M.EQ.0) THEN
                        SUMR = 2*SUMR - TOR
                        SUMI = 2*SUMI - TOI
        ELSE
                        SUMR = SUMR - TOR
                        SUMI = SUMI - TOI
        ENDIF
        GOTO 30
   24   IF (M.EQ.0) THEN
                        SUMR = 2*SUMR - TOI
                        SUMI = 2*SUMI + TOR
        ELSE
                        SUMR = SUMR - TOI
                        SUMI = SUMI + TOR
        ENDIF
   30 CONTINUE
c
c  calculate the normalizing factor by means of the identity
c    exp(-i*zargumt) = j(zargumt,0) + 2 * [ ((-i)**1) * j(zargumt,1) +
c                      ((-i)**2) * j(zargumt,2) + ((-i)**3) * j(zargumt,3) +...
c
      DJ = CDEXP((0.0D0,-1.0D0)*Z)/DCMPLX(SUMR,SUMI)
      JN = 1.0D0* (DJ * DCMPLX(JNR,JNI))
      J1 = 1.0D0* (DJ * DCMPLX(J1R,J1I))
      IF (CONJ) THEN
                      JN = DCONJG(JN)
                      J1 = DCONJG(J1)
      ENDIF
      IF (SIGN)              JN = -JN
      IF (SIGN.NEQV.RZNEG) J1 = -J1
      DJ = N*JN/ZZ - J1
      FLAG = .TRUE.
      RETURN
      END
```



```
C
      REAL*8 FUNCTION JB0(X)
C
C     berekent 0-de orde gewone reele Besselfunctie JB0 van reeel argument X
C       (t.b.v. matrix G: in dat geval aanroepen met X=j'_nu*r/a
C      waarbij  j'_nu is nu-de nulpunt van JB0', en a is pijpradius)
C
      implicit NONE
      COMPLEX*16 Z,J0,DJ,J1
      REAL*8 X
      LOGICAL FLAG
C
      Z = DCMPLX(X,0.0D0)
      CALL JB(Z,0,J0,DJ,J1,FLAG)
      IF(.NOT.FLAG) WRITE(*,*)' WAARSCHUWING: J0'
      JB0 = DBLE(J0)
C
      RETURN
      END
```

```
c
c ***************************************************************************
c
      SUBROUTINE KB(ZZ,NN,KN,DK,K1,FLAG)
c
c  in:
c     ZZ : complex argument
c     NN : order of besselfunction
c  out :
c     KN,DK,K1  :function values Kn, dKn, and Kn+1
c     FLAG  :indication if KB is computed correctly
c
      implicit NONE
      COMPLEX*16 Z,ZZ,KN,DK,K1,IPJN,IPJ1,DJ,CN,CN1
      REAL*8 PI,RE,RI,TR0,TI0,TR1,TI1,DIV,C0,C02,TPR,TPI
      INTEGER N,NN,JMB
      LOGICAL FLAG,SIGN,SZ,IMS
      DATA PI /3.141 592 653 589 793 238 462 643 383 279 D0/
      N = IABS(NN)
      Z = ZZ
c
c  flag indicates true or false (returned) KB-values
c
c  return if argument = 0
c
      KN = (0.0D0,0.0D0)
      FLAG = .FALSE.
      IF (Z.EQ.(0.0D0,0.0D0)) RETURN
c
c  if Re(Z) < 0. subroutine KZEONE is not valid.
c  KZEONE can be used when Z is modified to -Z, using
c  kn(z) = (-1)**n * kn(-z) - i* (-i)**n *pi*jn(-iz)  if   0 < arg(z) < pi
c  kn(z) = (-1)**n * kn(-z) + i* (-i)**n *pi*jn(-iz)  if -pi < arg(z) <= 0
c  where i=(0.,1.) and pi=3.14159....
c
      SZ = .FALSE.
      IF (DBLE(Z).LT.0.0D0) THEN
         SZ = .TRUE.
         IMS = .FALSE.
         IF (DIMAG(Z).LT.0.0D0) IMS = .TRUE.
         SIGN = .FALSE.
         IF (MOD(N,2).NE.0) SIGN = .TRUE.
         Z = -Z
      ENDIF
      RE = DBLE(Z)
      RI = DIMAG(Z)
c
c     subroutine KZEONE returns real and imaginary parts
c     of k(Z,0)*exp(Re(Z)) and k(Z,1)*exp(Re(Z))
c
      CALL KZEONE(RE,RI,TR0,TI0,TR1,TI1)
c
c     divide returned values by exp(re): multiply by exp(-re)
c
      DIV = DEXP(-RE)
      TR0 = TR0*DIV
      TI0 = TI0*DIV
      TR1 = TR1*DIV
      TI1 = TI1*DIV
c
c     k(z,0) and k(z,1) have been calculated now
      IF(N.GT.0) THEN
c
c     greater orders of k are calculated by means of the
c     forward recurrence relation
c
         C0 = 2./(RE*RE+RI*RI)
         DO 20 JMB  = 1,N
            C02 = JMB*C0
            TPR = C02*(RE*TR1+RI*TI1)+TR0
            TPI = C02*(RE*TI1-RI*TR1)+TI0
```



```
            TR0 = TR1
            TI0 = TI1
            TR1 = TPR
            TI1 = TPI
   20    CONTINUE
      ENDIF
c
c  if re(z) was positive or zero then the kn-value can be returned
c
      KN = DCMPLX(TR0,TI0)
      K1 = DCMPLX(TR1,TI1)
c
c  this part deals with k(z,n),  re(z)<0.
c  the formulas used are stated some lines back.
c
      IF (SZ) THEN
         CN = DCMPLX(0.0D0,1.0D0)
         CALL JB(CN*Z,N,IPJN,DJ,IPJ1,FLAG)
         IF(.NOT.FLAG) RETURN
c        calculate cn = -i*(-i)**n, cn1 = -i*(-i)**(n+1)
         GOTO(30,31,32,33) MOD(N,4)+1
 30       CN  = ( 0.0D0, -1.0D0)
          CN1 = ( -1.0D0, 0.0D0)
          GOTO 40
 31       CN  = ( -1.0D0, 0.0D0)
          CN1 = ( 0.0D0, 1.0D0)
          GOTO 40
 32       CN  = ( 0.0D0, 1.0D0)
          CN1 = ( 1.0D0, 0.0D0)
          GOTO 40
 33       CN  = ( 1.0D0, 0.0D0)
          CN1 = ( 0.0D0,-1.0D0)
 40       CONTINUE
          IPJN = CN *PI*IPJN
          IPJ1 = CN1*PI*IPJ1
          IF (IMS) THEN
                   IPJN = -IPJN
                   IPJ1 = -IPJ1
          ENDIF
          IF (SIGN) THEN
                  KN = -KN
          ELSE
                  K1 = -K1
          ENDIF
          KN = KN + IPJN
          K1 = K1 + IPJ1
      ENDIF
      DK = N*KN/ZZ - K1
      FLAG =.TRUE.
      RETURN
      END
```

```
c
c ***************************************************************************
c
      SUBROUTINE KZEONE(X,Y,RE0,IM0,RE1,IM1)
c
c  in : X,Y
c out : RE0,IM0,RE1,IM1
c
c  The variables X and Y are the real and imaginary parts of
c  the argument of the first two modified bessel functions of the
c  second kind, K0 and K1. RE0, IM0, RE1, and IM1 give
c  the real and imaginary parts of exp(X)*K0 and exp(X)*K1,
c  respectively. Although the real notation used in this
c  subroutine may seem inelegant when compared with the
c  complex notation that FORTRAN allows, this version runs
c  about 30% faster than one written using complex variables.
c
      implicit NONE
      DIMENSION EXSQ(8), TSQ(8)
      REAL*8 X,Y,RE0,RE1,IM0,IM1,RTERM,ITERM,EXSQ,TSQ,
     *       R1,R2,X2,Y2,P1,P2,T1,T2
      INTEGER N,L,M,K
      DATA TSQ(1) /0.0D0/, TSQ(2) /3.1930363392064D-1/, TSQ(3)
     *     /1.2907586229592D0/, TSQ(4) /2.9583744586967D0/, TSQ(5)
     *     /5.4090315972444D0/, TSQ(6) /8.8040795780568D0/, TSQ(7)
     *     /1.3468535743252D1/, TSQ(8) /2.0249916365871D1/, EXSQ(1)
     *     /0.5641003087264D0/, EXSQ(2) /0.4120286874989D0/, EXSQ(3)
     *     /0.1584889157959D0/, EXSQ(4) /0.3078003387255D-1/, EXSQ(5)
     *     /0.2778068842913D-2/, EXSQ(6) /0.1000044412325D-3/,
     *     EXSQ(7) /0.1059115547711D-5/, EXSQ(8) /0.1522475804254D-8/
c
c  the arrays TSQ and EXSQ contain the square of the
c  abcisses and the weight factors used in the gauss-
c  hermite quadrature.
c
      R2 = X*X+Y*Y
      IF (R2.GE.1.96D2) GOTO 40
      IF (R2.GE.1.849D1) GOTO 20
c  this section calculates the functions using the series expansions
        X2 = X/2.0D0
        Y2 = Y/2.0D0
        P1 = X2*X2
        P2 = Y2*Y2
        T1 = -DLOG(P1+P2)/2.0D0 - 0.577 215 664 901 532 860 606 512 D0
c the constant in the preceding statement is Euler's constant
        T2 = -DATAN2(Y,X)
        X2 = P1-P2
        Y2 = X*Y2
        RTERM = 1.0D0
        ITERM = 0.0D0
        RE0 = T1
        IM0 = T2
        T1 = T1+0.5D0
        RE1 = T1
        IM1 = T2
        P2 = DSQRT(R2)
        L = INT(2.106D0 * P2 + 4.4D0)
        IF(P2.LT.8.0D-1) L = INT(2.129D0 * P2 + 4.0D0)
        DO 10 N = 1,L
           P1 = N
           P2 = N*N
           R1 = RTERM
           RTERM = (R1*X2-ITERM*Y2)/P2
           ITERM = (R1*Y2+ITERM*X2)/P2
           T1 = T1+0.5D0/P1
           RE0 = RE0+T1*RTERM-T2*ITERM
           IM0 = IM0+T1*ITERM+T2*RTERM
           P1 = P1+1.0D0
           T1 = T1+0.5D0/P1
           RE1 = RE1+(T1*RTERM-T2*ITERM)/P1
           IM1 = IM1+(T1*ITERM+T2*RTERM)/P1
```



```
   10    CONTINUE
         R1 =  X/R2-0.5D0*(X*RE1-Y*IM1)
         R2 = -Y/R2-0.5D0*(X*IM1+Y*RE1)
         P1 = DEXP(X)
         RE0 = P1*RE0
         IM0 = P1*IM0
         RE1 = P1*R1
         IM1 = P1*R2
         RETURN
c
c  this section calculates the functions using the integral
c  representation, eqn 3, evaluated with 15 point gauss-hermite quadrature
c
   20 X2 = 2.0D0*X
      Y2 = 2.0D0*Y
      R1 = Y2*Y2
      P1 = DSQRT(X2*X2+R1)
      P2 = DSQRT(P1+X2)
      T1 = EXSQ(1)/(2.0D0*P1)
      RE0 = T1*P2
      IM0 = T1/P2
      RE1 = 0.0D0
      IM1 = 0.0D0
      DO 30 N = 2,8
         T2 = X2+TSQ(N)
         P1 = DSQRT(T2*T2+R1)
         P2 = DSQRT(P1+T2)
         T1 = EXSQ(N)/P1
         RE0 = RE0+T1*P2
         IM0 = IM0+T1/P2
         T1 = EXSQ(N)*TSQ(N)
         RE1 = RE1+T1*P2
         IM1 = IM1+T1/P2
   30 CONTINUE
      T2=-Y2*IM0
      RE1 = RE1/R2
      R2 = Y2*IM1/R2
      RTERM =  1.414213562373O9D0*DCOS(Y)
      ITERM = -1.414213562373O9D0*DSIN(Y)
c the constant in the previous statements is of course sqrt(2)
      IM0 = RE0*ITERM+T2*RTERM
      RE0 = RE0*RTERM-T2*ITERM
      T1 = RE1*RTERM-R2*ITERM
      T2 = RE1*ITERM+R2*RTERM
      RE1 = T1*X+T2*Y
      IM1 = -T1*Y+T2*X
      RETURN
c  this section calculates the functions using the asymptotic
c  expansions
   40 RTERM = 1.0D0
      ITERM = 0.0D0
      RE0 = 1.0D0
      IM0 = 0.0D0
      RE1 = 1.0D0
      IM1 = 0.0D0
      P1 = 8.0D0*R2
      P2 = DSQRT(R2)
      L = INT(3.91D0+8.12D1/P2)
      R1 = 1.0D0
      R2 = 1.0D0
      M = -8
      K = 3
      DO 50 N = 1,L
         M = M+8
         K = K-M
         R1 = DBLE(K-4)*R1
         R2 = DBLE(K)*R2
         T1 = DBLE(N)*P1
         T2 = RTERM
         RTERM = ( T2*X+ITERM*Y)/T1
         ITERM = (-T2*Y+ITERM*X)/T1
```

```
          RE0 = RE0+R1*RTERM
          IM0 = IM0+R1*ITERM
          RE1 = RE1+R2*RTERM
          IM1 = IM1+R2*ITERM
   50   CONTINUE
        T1 = DSQRT(P2+X)
        T2 = -Y/T1
        P1 = 1.77245 38509 05516 02730 D0/2/P2
c       this constant is sqrt(pi)
        RTERM =  P1*DCOS(Y)
        ITERM = -P1*DSIN(Y)
        R1 = RE0*RTERM-IM0*ITERM
        R2 = RE0*ITERM+IM0*RTERM
        RE0 = T1*R1-T2*R2
        IM0 = T1*R2+T2*R1
        R1 = RE1*RTERM-IM1*ITERM
        R2 = RE1*ITERM+IM1*RTERM
        RE1 = T1*R1-T2*R2
        IM1 = T1*R2+T2*R1
        RETURN
        END
```



```
C
C ***************************************************************************
C
      SUBROUTINE LOGO
C
      WRITE(*,'(1H4)')
      WRITE(*,910)'2DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD?'
      WRITE(*,910)'3   Welkom bij het acoustic ranging programma     3'
      WRITE(*,910)'3                                                 3'
      WRITE(*,910)'3                        RANGE                    3'
      WRITE(*,910)'3                                                 3'
      WRITE(*,910)'3 Instituut Wiskundige Dienstverlening Eindhoven 3'
      WRITE(*,910)'3      Technische Universiteit Eindhoven          3'
      WRITE(*,910)'@DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDY'
      WRITE(*,'(1H9)')
      RETURN
910   FORMAT(T14,A50)
      END
```

```
C
C*********************************************************************
C
      SUBROUTINE NEWT(X,F)
      implicit NONE
      COMPLEX*16 Z,JM,DJ,J1
      REAL*8 X,F,DIF
      LOGICAL FLAG
C
C **  De functie berekent met Newton iteratie een nulpunt X van de
C **  afgeleide Besselfunctie J0'=-J1 (bij in is X een startwaarde),
C **  en het bijbehorende F=J0(X)=J1'(X)
C
      DIF = 0D0
 10   CONTINUE
        X = X - DIF
        Z = DCMPLX(X,0.0D0)
        CALL JB(Z,1,JM,DJ,J1,FLAG)
        IF(.NOT.FLAG) WRITE(*,*)' WAARSCHUWING: JB NEWT'
        DIF = DBLE(JM)/DBLE(DJ)
      IF (DABS(DIF).GT.(1.0D-12)*DABS(X)) GOTO 10
      F = DBLE(DJ)
      RETURN
      END
```



```
C
C
      double precision function pythag(a,b)
      implicit NONE
      double precision a,b
c
c     finds dsqrt(a**2+b**2) without overflow or destructive underflow
c
      double precision p,r,s,t,u
      p = dmax1(dabs(a),dabs(b))
      if (p .eq. 0.0d0) go to 20
      r = (dmin1(dabs(a),dabs(b))/p)**2
   10 continue
         t = 4.0d0 + r
         if (t .eq. 4.0d0) go to 20
         s = r/t
         u = 1.0d0 + 2.0d0*s
         p = u*p
         r = (s/u)**2 * r
      go to 10
   20 pythag = p
      return
      end
```

```
C
C*********************************************************************
C
      SUBROUTINE ROMBRG(F,A,B,EPS,MAXITER,FLAG,ROMINT)
      implicit NONE
      INTEGER ZAT
      PARAMETER (ZAT=16)
      REAL*8 F,R(0:1,0:ZAT),SM,D,ROMINT,A,B,H,EPS,ERR,FR
      INTEGER MAXITER,N,M,K
      LOGICAL FLAG
      EXTERNAL F
C
      IF(MAXITER.GT.ZAT) STOP 'ROMBRG'
      N = 0
      M = 1
      H = B-A
      R(1,0) = (F(A)+F(B))*H/2.0D0
 10   CONTINUE
      IF (N.LE.MAXITER) THEN
         N  = N+1
         H  = H/2.0D0
         SM = 0
         DO 20 K=0,N-1
            R(0,K) = R(1,K)
 20      CONTINUE
         DO 30 K=1,M
            SM = SM + F(A+(2*K-1)*H)
 30      CONTINUE
         R(1,0) = R(0,0)/2.0D0 + H*SM
         M  = 2*M
         FR = 1.D0
         DO 40 K=1,N
            FR = FR*4.0D0
            D  = R(1,K-1)-R(0,K-1)
            R(1,K) = R(1,K-1) + D/(FR-1)
            IF(R(1,K).NE.0.0D0) THEN
              ERR = DABS(D/R(1,K))/FR
            ELSE
              ERR = DABS(D)/FR
            ENDIF
            IF ((ERR.LT.EPS).AND.(N.GT.3)) THEN
                ROMINT = R(1,K)
                FLAG = .TRUE.
                RETURN
            ENDIF
 40      CONTINUE
      ELSE
         ROMINT = 0.0D0
         FLAG = .FALSE.
         RETURN
      ENDIF
      GOTO 10
      END
```



```
      subroutine RWPSOL(x,h,n,consta,ierr,mumax,gamma,psi,ldapsi)
      implicit NONE
C
C     3 april  '92
C
      integer n,ierr,nm,i,mumax,j, mi1, mi2, mmm, ldapsi
      parameter(nm=1001, mmm=100)
C
C ldapsi is the leading dimension of the array psi
C
      double precision x,consta,gamma1(mmm),gamma2(mmm),psi1(nm,mmm),
     +                 psi2(nm,mmm), mpsi, mpsio,h,psi(ldapsi,mumax)
      double complex gamma(mumax)
C
      external EIGEN
C
      call EIGEN(x, h, n, consta,ierr,mumax, gamma1, psi1)
      call EIGEN(x, h, 2*n, consta,  ierr, mumax,gamma2,psi2)
C
C
C   h**2-extraplolatie
C
      do 15 j=1,mumax
         gamma(j)=dcmplx(0.0d0,-1.0d0)*
     +          sqrt(dcmplx(gamma2(j)+(gamma2(j)-gamma1(j))/3.0d0))
         mpsi=abs(psi1(1,j))
         mi1=1
         do 13 i=1,n+1
               mpsio=mpsi
               mpsi=max(mpsi,abs(psi1(i,j)))
               if(mpsio.ne.mpsi) mi1=i
 13      continue
         mi2=1
         mpsi=abs(psi2(1,j))
         do 16 i=1, n+1
               mpsio=mpsi
               mpsi=max(mpsi,abs(psi2(2*i-1,j)))
               if(mpsio.ne.mpsi) mi2=2*i-1
 16      continue
         if(sign(1,psi1(mi1,j)).ne.sign(1,psi2(mi2,j))) then
              do 17 i=1,n+1
                 psi1(i,j)=-psi1(i,j)
 17           continue
         endif
         do 14 i=1,n+1
            psi(i,j)=psi2(2*i-1,j)+(psi2(2*i-1,j)-psi1(i,j))/3.0d0
 14      continue
 15   continue
      return
      end
```

```
C
C*******************************************************************8
C
      REAL*8 FUNCTION TGRND1(X)
C
      implicit NONE
      INTEGER MMM,IK1TAB
      PARAMETER (MMM=30,IK1TAB=1025)
C
      REAL*8  PI,K0A,EPSINT,ZJ1(MMM),ZJ2(MMM),JBZ(MMM)
      INTEGER MU,NU,TELMAX(3)
      LOGICAL EERSTE
C
      COMMON//EPSINT,MU,NU,K0A,ZJ1,ZJ2,JBZ,EERSTE,TELMAX,PI
C
      COMPLEX*16 KM,IM,DF,F1,Z
      REAL*8 IK1(IK1TAB),X
      INTEGER TEL
      LOGICAL FLAG
      SAVE IK1,TEL
C
      IF(EERSTE) TEL = 0
      EERSTE = .FALSE.
      TEL = TEL+1
      IF(TEL.GT.TELMAX(1)) THEN
        IF(X.EQ.0D0) THEN
          IK1(TEL) = 0D0
        ELSE
          Z = DCMPLX(X,0D0)
          CALL KB(Z,1,KM,DF,F1,FLAG)
          IF (.NOT.FLAG) WRITE(*,*)' WAARSCHUWING: TGRND1 KB'
          CALL IB(Z,1,IM,DF,F1,FLAG)
          IF (.NOT.FLAG) WRITE(*,*)' WAARSCHUWING: TGRND1 IB'
          IK1(TEL) = DBLE(IM)*DBLE(KM)*X*X*X/DSQRT(K0A*K0A+X*X)
        ENDIF
        TELMAX(1) = TEL
      ENDIF
      IF (X.EQ.0D0) THEN
        TGRND1 = 0D0
      ELSE
        IF ((MU.NE.1).OR.(NU.NE.1)) THEN
          TGRND1 = IK1(TEL)/(X*X+ZJ2(MU))/(X*X+ZJ2(NU))
        ELSE
          TGRND1 = (DBLE(IM)*DBLE(KM)-0.5D0)/X/DSQRT(K0A*K0A+X*X)
        ENDIF
      ENDIF
      RETURN
      END
```



```
C
C*******************************************************************8
C
      REAL*8 FUNCTION TGRND2(X)
C
      implicit NONE
      INTEGER MMM,IK2TAB
      PARAMETER (MMM=30,IK2TAB=1025)
C
      REAL*8  PI,K0A,EPSINT,ZJ1(MMM),ZJ2(MMM),JBZ(MMM)
      INTEGER MU,NU,TELMAX(3)
      LOGICAL EERSTE
C
      COMMON//EPSINT,MU,NU,K0A,ZJ1,ZJ2,JBZ,EERSTE,TELMAX,PI
C
      COMPLEX*16 Z,KM,IM,DF,F1
      REAL*8 IK2(IK2TAB),X,X2
      INTEGER TEL
      LOGICAL FLAG
      SAVE IK2,TEL
C
      IF(EERSTE) TEL = 0
      EERSTE = .FALSE.
      TEL = TEL+1
      IF(TEL.GT.TELMAX(2)) THEN
        IF(X.EQ.0D0) THEN
          IK2(TEL) = 0D0
        ELSE
          IF (X.GT.1.6D-2) THEN
            Z = DCMPLX(1/X,0D0)
            CALL KB(Z,1,KM,DF,F1,FLAG)
            IF(.NOT.FLAG)WRITE(*,*)' WAARSCHUWING: TGRND2 KB'
            CALL IB(Z,1,IM,DF,F1,FLAG)
            IF(.NOT.FLAG)WRITE(*,*)' WAARSCHUWING: TGRND2 IB'
            IK2(TEL) = DBLE(IM)*DBLE(KM)
          ELSE
            X2 = X*X/4
            IK2(TEL) = 0.5D0*X*(1D0-1.5D0*X2*(1D0+3.75D0*X2*
     >                 (1D0+17.5D0*X2)))
          ENDIF
          IK2(TEL) = IK2(TEL)/DSQRT(1+K0A*K0A*X*X)
        ENDIF
        TELMAX(2) = TEL
      ENDIF
      IF (X.EQ.0D0) THEN
        TGRND2 = 0D0
      ELSE
        IF ((MU.NE.1).OR.(NU.NE.1)) THEN
          TGRND2 = IK2(TEL)/(1+X*X*ZJ2(MU))/(1+X*X*ZJ2(NU))
        ELSE
          TGRND2 = IK2(TEL)
        ENDIF
      ENDIF
      RETURN
      END
```

```
C
C*********************************************************************
C
      COMPLEX*16 FUNCTION TGRND3(X)
C
      implicit NONE
      INTEGER MMM,IK3TAB
      PARAMETER (MMM=30,IK3TAB=1025)
C
      REAL*8  PI,K0A,EPSINT,ZJ1(MMM),ZJ2(MMM),JBZ(MMM)
      INTEGER MU,NU,TELMAX(3)
      LOGICAL EERSTE
C
      COMMON//EPSINT,MU,NU,K0A,ZJ1,ZJ2,JBZ,EERSTE,TELMAX,PI
C
      COMPLEX*16 IK3(IK3TAB),H2,DF,F1,Z
      REAL*8 X,T,JM
      INTEGER TEL
      LOGICAL FLAG
      SAVE IK3,TEL
C
      T = K0A*DSIN(X)
      IF(EERSTE) TEL = 0
      EERSTE = .FALSE.
      TEL = TEL+1
      IF(TEL.GT.TELMAX(3)) THEN
        IF(X.EQ.0D0) THEN
          IK3(TEL) = (0D0,1D0)
        ELSE
          Z = DCMPLX(T,0D0)
          CALL H2B(Z,1,H2,DF,F1,FLAG)
          IF(.NOT.FLAG) WRITE(*,*)' WAARSCHUWING: TGRND3 H2B'
          JM = DBLE(H2)
          IK3(TEL) = T*T*T*JM*H2
        ENDIF
        TELMAX(3) = TEL
      ENDIF
      IF (X.EQ.0D0) THEN
        TGRND3 = (0D0,0D0)
      ELSE
        IF ((MU.NE.1).OR.(NU.NE.1)) THEN
           TGRND3 = IK3(TEL)/(T*T-ZJ2(MU))/(T*T-ZJ2(NU))
        ELSE
           TGRND3 = (JM*H2 - (0D0,1D0)/PI)/T
        ENDIF
      ENDIF
      RETURN
      END
```



```
      subroutine tsturm(nm,n,eps1,d,e,e2,lb,ub,mm,m,w,z,
     x                  ierr,rv1,rv2,rv3,rv4,rv5,rv6)
c
      implicit NONE
      integer i,j,k,m,n,p,q,r,s,ii,ip,jj,mm,m1,m2,nm,its,
     x        ierr,group,isturm
      double precision d(n),e(n),e2(n),w(mm),z(nm,mm),
     x        rv1(n),rv2(n),rv3(n),rv4(n),rv5(n),rv6(n)
      double precision u,v,lb,t1,t2,ub,uk,xu,x0,x1,eps1,eps2,eps3,eps4,
     x        norm,tst1,tst2,epslon,pythag, machep
c
c     this subroutine is a translation of the algol procedure tristurm
c     by peters and wilkinson.
c     handbook for auto. comp., vol.ii-linear algebra, 418-439(1971).
c
c     this subroutine finds those eigenvalues of a tridiagonal
c     symmetric matrix which lie in a specified interval and their
c     associated eigenvectors, using bisection and inverse iteration.
c
c     on input
c
c        nm must be set to the row dimension of two-dimensional
c          array parameters as declared in the calling program
c          dimension statement.
c
c        n is the order of the matrix.
c
c        eps1 is an absolute error tolerance for the computed
c          eigenvalues.  it should be chosen commensurate with
c          relative perturbations in the matrix elements of the
c          order of the relative machine precision.  if the
c          input eps1 is non-positive, it is reset for each
c          submatrix to a default value, namely, minus the
c          product of the relative machine precision and the
c          1-norm of the submatrix.
c
c        d contains the diagonal elements of the input matrix.
c
c        e contains the subdiagonal elements of the input matrix
c          in its last n-1 positions.  e(1) is arbitrary.
c
c        e2 contains the squares of the corresponding elements of e.
c          e2(1) is arbitrary.
c
c        lb and ub define the interval to be searched for eigenvalues.
c          if lb is not less than ub, no eigenvalues will be found.
c
c        mm should be set to an upper bound for the number of
c          eigenvalues in the interval.  warning. if more than
c          mm eigenvalues are determined to lie in the interval,
c          an error return is made with no values or vectors found.
c
c     on output
c
c        eps1 is unaltered unless it has been reset to its
c          (last) default value.
c
c        d and e are unaltered.
c
c        elements of e2, corresponding to elements of e regarded
c          as negligible, have been replaced by zero causing the
c          matrix to split into a direct sum of submatrices.
c          e2(1) is also set to zero.
c
c        m is the number of eigenvalues determined to lie in (lb,ub).
c
c        w contains the m eigenvalues in ascending order if the matrix
c          does not split.  if the matrix splits, the eigenvalues are
c          in ascending order for each submatrix.  if a vector error
c          exit is made, w contains those values already found.
c
```

```
c         z contains the associated set of orthonormal eigenvectors.
c           if an error exit is made, z contains those vectors
c           already found.
c
c         ierr is set to
c           zero       for normal return,
c           3*n+1      if m exceeds mm.
c           4*n+r      if the eigenvector corresponding to the r-th
c                      eigenvalue fails to converge in 5 iterations.
c
c         rv1, rv2, rv3, rv4, rv5, and rv6 are temporary storage arrays.
c
c     the algol procedure sturmcnt contained in tristurm
c     appears in tsturm in-line.
c
c     calls pythag for  dsqrt(a*a + b*b) .
c
c     questions and comments should be directed to burton s. garbow,
c     mathematics and computer science div, argonne national laboratory
c
c     this version dated august 1983.
c
c     ------------------------------------------------------------------
c
      machep=1.0d0
1234  machep=machep/2.0d0
      if(machep+1.0d0.ne.1.0d0) goto 1234
      machep=machep*2.0d0
c
      ierr = 0
      t1 = lb
      t2 = ub
c     ......... look for small sub-diagonal entries .........
      do 40 i = 1, n
         if (i .eq. 1) go to 20
         tst1 = dabs(d(i)) + dabs(d(i-1))
         tst2 = tst1 + dabs(e(i))
         if (tst2 .gt. tst1) go to 40
   20    e2(i) = 0.0d0
   40 continue
c     ......... determine the number of eigenvalues
c                in the interval .........
      p = 1
      q = n
      x1 = ub
      isturm = 1
      go to 320
   60 m = s
      x1 = lb
      isturm = 2
      go to 320
   80 m = m - s
      if (m .gt. mm) go to 980
      q = 0
      r = 0
c     ......... establish and process next submatrix, refining
c                interval by the gerschgorin bounds .........
  100 if (r .eq. m) go to 1001
      p = q + 1
      xu = d(p)
      x0 = d(p)
      u = 0.0d0
c
      do 120 q = p, n
         x1 = u
         u = 0.0d0
         v = 0.0d0
         if (q .eq. n) go to 110
         u = dabs(e(q+1))
         v = e2(q+1)
  110    xu = dmin1(d(q)-(x1+u),xu)
```



```
         x0 = dmax1(d(q)+(x1+u),x0)
         if (v .eq. 0.0d0) go to 140
  120 continue
c
  140 x1 = epslon(dmax1(dabs(xu),dabs(x0)),machep)
      if (eps1 .le. 0.0d0) eps1 = -x1
      if (p .ne. q) go to 180
c     ......... check for isolated root within interval .........
      if (t1 .gt. d(p) .or. d(p) .ge. t2) go to 940
      r = r + 1
c
      do 160 i = 1, n
  160 z(i,r) = 0.0d0
c
      w(r) = d(p)
      z(p,r) = 1.0d0
      go to 940
  180 u = q-p+1
      x1 = u * x1
      lb = dmax1(t1,xu-x1)
      ub = dmin1(t2,x0+x1)
      x1 = lb
      isturm = 3
      go to 320
  200 m1 = s + 1
      x1 = ub
      isturm = 4
      go to 320
  220 m2 = s
      if (m1 .gt. m2) go to 940
c     ......... find roots by bisection .........
      x0 = ub
      isturm = 5
c
      do 240 i = m1, m2
         rv5(i) = ub
         rv4(i) = lb
  240 continue
c     ......... loop for k-th eigenvalue
c                for k=m2 step -1 until m1 do --
c                (-do- not used to legalize -computed go to-) .........
      k = m2
  250    xu = lb
c     ......... for i=k step -1 until m1 do -- .........
         do 260 ii = m1, k
            i = m1 + k - ii
            if (xu .ge. rv4(i)) go to 260
            xu = rv4(i)
            go to 280
  260    continue
c
  280    if (x0 .gt. rv5(k)) x0 = rv5(k)
c     ......... next bisection step .........
  300    x1 = (xu + x0) * 0.5d0
         if ((x0 - xu) .le. dabs(eps1)) go to 420
         tst1 = 2.0d0 * (dabs(xu) + dabs(x0))
         tst2 = tst1 + (x0 - xu)
         if (tst2 .eq. tst1) go to 420
c     ......... in-line procedure for sturm sequence .........
  320    s = p - 1
         u = 1.0d0
c
         do 340 i = p, q
            if (u .ne. 0.0d0) go to 325
            v = dabs(e(i)) / epslon(1.0d0, machep)
            if (e2(i) .eq. 0.0d0) v = 0.0d0
            go to 330
  325       v = e2(i) / u
  330       u = d(i) - x1 - v
            if (u .lt. 0.0d0) s = s + 1
  340    continue
```

```
c
      go to (60,80,200,220,360), isturm
c     ......... refine intervals .........
  360 if (s .ge. k) go to 400
      xu = x1
      if (s .ge. m1) go to 380
      rv4(m1) = x1
      go to 300
  380 rv4(s+1) = x1
      if (rv5(s) .gt. x1) rv5(s) = x1
      go to 300
  400 x0 = x1
      go to 300
c     ......... k-th eigenvalue found .........
  420 rv5(k) = x1
      k = k - 1
      if (k .ge. m1) go to 250
c     ......... find vectors by inverse iteration .........
      norm = dabs(d(p))
      ip = p + 1
c
      do 500 i = ip, q
  500 norm = dmax1(norm, dabs(d(i)) + dabs(e(i)))
c     ......... eps2 is the criterion for grouping,
c               eps3 replaces zero pivots and equal
c               roots are modified by eps3,
c               eps4 is taken very small to avoid overflow .........
      eps2 = 1.0d-3 * norm
      eps3 = epslon(norm,machep)
      uk = q - p + 1
      eps4 = uk * eps3
      uk = eps4 / dsqrt(uk)
      group = 0
      s = p
c
      do 920 k = m1, m2
         r = r + 1
         its = 1
         w(r) = rv5(k)
         x1 = rv5(k)
c     ......... look for close or coincident roots .........
         if (k .eq. m1) go to 520
         if (x1 - x0 .ge. eps2) group = -1
         group = group + 1
         if (x1 .le. x0) x1 = x0 + eps3
c     ......... elimination with interchanges and
c               initialization of vector .........
  520    v = 0.0d0
c
         do 580 i = p, q
            rv6(i) = uk
            if (i .eq. p) go to 560
            if (dabs(e(i)) .lt. dabs(u)) go to 540
            xu = u / e(i)
            rv4(i) = xu
            rv1(i-1) = e(i)
            rv2(i-1) = d(i) - x1
            rv3(i-1) = 0.0d0
            if (i .ne. q) rv3(i-1) = e(i+1)
            u = v - xu * rv2(i-1)
            v = -xu * rv3(i-1)
            go to 580
  540       xu = e(i) / u
            rv4(i) = xu
            rv1(i-1) = u
            rv2(i-1) = v
            rv3(i-1) = 0.0d0
  560       u = d(i) - x1 - xu * v
            if (i .ne. q) v = e(i+1)
  580    continue
c
```



```
         if (u .eq. 0.0d0) u = eps3
         rv1(q) = u
         rv2(q) = 0.0d0
         rv3(q) = 0.0d0
c     ......... back substitution
c                for i=q step -1 until p do -- .........
  600    do 620 ii = p, q
            i = p + q - ii
            rv6(i) = (rv6(i) - u * rv2(i) - v * rv3(i)) / rv1(i)
            v = u
            u = rv6(i)
  620    continue
c     ......... orthogonalize with respect to previous
c                members of group .........
         if (group .eq. 0) go to 700
c
         do 680 jj = 1, group
            j = r - group - 1 + jj
            xu = 0.0d0
c
            do 640 i = p, q
  640       xu = xu + rv6(i) * z(i,j)
c
            do 660 i = p, q
  660       rv6(i) = rv6(i) - xu * z(i,j)
c
  680    continue
c
  700    norm = 0.0d0
c
         do 720 i = p, q
  720    norm = norm + dabs(rv6(i))
c
         if (norm .ge. 1.0d0) go to 840
c     ......... forward substitution .........
         if (its .eq. 5) go to 960
         if (norm .ne. 0.0d0) go to 740
         rv6(s) = eps4
         s = s + 1
         if (s .gt. q) s = p
         go to 780
  740    xu = eps4 / norm
c
         do 760 i = p, q
  760    rv6(i) = rv6(i) * xu
c     ......... elimination operations on next vector
c                iterate .........
  780    do 820 i = ip, q
            u = rv6(i)
c     ......... if rv1(i-1) .eq. e(i), a row interchange
c                was performed earlier in the
c                triangularization process .........
            if (rv1(i-1) .ne. e(i)) go to 800
            u = rv6(i-1)
            rv6(i-1) = rv6(i)
  800       rv6(i) = u - rv4(i) * rv6(i-1)
  820    continue
c
         its = its + 1
         go to 600
c     ......... normalize so that sum of squares is
c                1 and expand to full order .........
  840    u = 0.0d0
c
         do 860 i = p, q
  860    u = pythag(u,rv6(i))
c
         xu = 1.0d0 / u
c
         do 880 i = 1, n
  880    z(i,r) = 0.0d0
```

```
c
         do 900 i = p, q
  900    z(i,r) = rv6(i) * xu
c
         x0 = x1
  920 continue
c
  940 if (q .lt. n) go to 100
      go to 1001
c     ......... set error -- non-converged eigenvector .........
  960 ierr = 4 * n + r
      go to 1001
c     ......... set error -- underestimate of number of
c                 eigenvalues in interval .........
  980 ierr = 3 * n + 1
 1001 lb = t1
      ub = t2
      return
      end
```



```
c
c ***********************************************************************
c
      SUBROUTINE YB(ZZ,NN,YN,DY,Y1,FLAG)
c     in: ZZ  : complex argument
c         NN  : order of bessel functions
c     out: YN,DY,Y1: bessel function values Yn, dYn, Yn+1
c
c     this subroutine calculates bessel functions Y of the
c     second kind of order N and with complex argument
c
      implicit NONE
      COMPLEX*16 Z,ZZ,YN,DY,Y1,CN,FI,KN,DK,K1,JN,DJ,J1
      REAL*8 PI
      INTEGER N,NN
      LOGICAL FLAG
      DATA PI /3.141 592 653 589 793 238 462 643 383 279 D0/
      Z = ZZ
      N = NN
      FLAG =.FALSE.
      YN = (0.0D0,0.0D0)
c
c     flag is used to indicate true or false returned YB-values.
c     return if argument is 0.
c
      IF(Z.EQ.(0.0D0,0.0D0)) RETURN
c
c     the yb function values are calculated by means of the following
c     formulas:
c     y(z,n)= i*j(z,n)-2*(-i)**n*k(-i*z,n)/pi   ( 0 < arg z <= pi )
c     y(z,n)=-i*j(z,n)-2*i**n*k(i*z,n)/pi       (-pi< arg z <= 0 )
c     if (im(z)>0) or (im(z)=0 and re(z)<0) then
c     the first formula is used, otherwise the last
c
      CALL JB(Z,N,JN,DJ,J1,FLAG)
c
c    not inefficient because jn is not calculated in kb
c
      IF (.NOT. FLAG) RETURN
      IF ((DIMAG(Z).GT.0.0D0).OR.((DIMAG(Z).EQ.0.0D0).AND.
     >                                    (DBLE(Z).LT.0.0D0))) THEN
         FI = (0.0D0,-1.0D0)
      ELSE
         FI = (0.0D0, 1.0D0)
      ENDIF
      CALL KB(FI*Z,N,KN,DK,K1,FLAG)
      IF(.NOT.FLAG) RETURN
c
c     calculate (+/-i)**n
c
      GOTO (10,11,12,13) MOD(N,4)+1
 10    CN = ( 1.0D0, 0.0D0)
       GOTO 20
 11    CN = FI
       GOTO 20
 12    CN = (-1.0D0, 0.0D0)
       GOTO 20
 13    CN = -FI
 20   CONTINUE
c
c     calculates yb(..)
c
      YN = -FI*JN - 2.0D0*   CN*KN/PI
      Y1 = -FI*J1 - 2.0D0*FI*CN*K1/PI
      DY = N*YN/Z - Y1
      FLAG = .TRUE.
      RETURN
      END
```

```
      double precision function dcabs1(z)
      double complex z,zz
      double precision t(2)
      equivalence (zz,t(1))
      zz = z
      dcabs1 = dabs(t(1)) + dabs(t(2))
      return
      end
```



```
      double precision function dzasum(n,zx,incx)
c
c     takes the sum of the absolute values.
c     jack dongarra, 3/11/78.
c
      double complex zx(1)
      double precision stemp,dcabs1
      integer n,incx,ix,i
c
      dzasum = 0.0d0
      stemp = 0.0d0
      if(n.le.0)return
      if(incx.eq.1)go to 20
c
c        code for increments not equal to 1
c
      ix = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      do 10 i = 1,n
        stemp = stemp + dcabs1(zx(ix))
        ix = ix + incx
   10 continue
      dzasum = stemp
      return
c
c        code for increments equal to 1
c
   20 do 30 i = 1,n
        stemp = stemp + dcabs1(zx(i))
   30 continue
      dzasum = stemp
      return
      end
```

```
      integer function izamax(n,zx,incx)
c
c     finds the index of element having max. absolute value.
c     jack dongarra, 1/15/85.
c
      double complex zx(1)
      double precision smax
      integer i,incx,ix,n
      double precision dcabs1
c
      izamax = 0
      if(n.lt.1)return
      izamax = 1
      if(n.eq.1)return
      if(incx.eq.1)go to 20
c
c        code for increment not equal to 1
c
      ix = 1
      smax = dcabs1(zx(1))
      ix = ix + incx
      do 10 i = 2,n
         if(dcabs1(zx(ix)).le.smax) go to 5
         izamax = i
         smax = dcabs1(zx(ix))
    5    ix = ix + incx
   10 continue
      return
c
c        code for increment equal to 1
c
   20 smax = dcabs1(zx(1))
      do 30 i = 2,n
         if(dcabs1(zx(i)).le.smax) go to 30
         izamax = i
         smax = dcabs1(zx(i))
   30 continue
      return
      end
```



```
      LOGICAL FUNCTION LSAME ( CA, CB )
*     .. Scalar Arguments ..
      CHARACTER*1            CA, CB
*     ..
*
*  Purpose
*  =======
*
*  LSAME  tests if CA is the same letter as CB regardless of case.
*  CB is assumed to be an upper case letter. LSAME returns .TRUE. if
*  CA is either the same as CB or the equivalent lower case letter.
*
*  N.B. This version of the routine is only correct for ASCII code.
*       Installers must modify the routine for other character-codes.
*
*       For EBCDIC systems the constant IOFF must be changed to -64.
*       For CDC systems using 6-12 bit representations, the system-
*       specific code in comments must be activated.
*
*  Parameters
*  ==========
*
*  CA     - CHARACTER*1
*  CB     - CHARACTER*1
*           On entry, CA and CB specify characters to be compared.
*           Unchanged on exit.
*
*
*  Auxiliary routine for Level 2 Blas.
*
*  -- Written on 20-July-1986
*     Richard Hanson, Sandia National Labs.
*     Jeremy Du Croz, Nag Central Office.
*
*     .. Parameters ..
      INTEGER                IOFF
      PARAMETER            ( IOFF=32 )
*     .. Intrinsic Functions ..
      INTRINSIC              ICHAR
*     .. Executable Statements ..
*
*     Test if the characters are equal
*
      LSAME = CA .EQ. CB
*
*     Now test for equivalence
*
      IF ( .NOT.LSAME ) THEN
         LSAME = ICHAR(CA) - IOFF .EQ. ICHAR(CB)
      END IF
*
      RETURN
*
*  The following comments contain code for CDC systems using 6-12 bit
*  representations.
*
*     .. Parameters ..
*     INTEGER                ICIRFX
*     PARAMETER            ( ICIRFX=62 )
*     .. Scalar Arguments ..
*     CHARACTER*1            CB
*     .. Array Arguments ..
*     CHARACTER*1            CA(*)
*     .. Local Scalars ..
*     INTEGER                IVAL
*     .. Intrinsic Functions ..
*     INTRINSIC              ICHAR, CHAR
*     .. Executable Statements ..
*
*     See if the first character in string CA equals string CB.
*
```

```
*     LSAME = CA(1) .EQ. CB .AND. CA(1) .NE. CHAR(ICIRFX)
*
*     IF (LSAME) RETURN
*
*     The characters are not identical. Now check them for equivalence.
*     Look for the 'escape' character, circumflex, followed by the
*     letter.
*
*     IVAL = ICHAR(CA(2))
*     IF (IVAL.GE.ICHAR('A') .AND. IVAL.LE.ICHAR('Z')) THEN
*        LSAME = CA(1) .EQ. CHAR(ICIRFX) .AND. CA(2) .EQ. CB
*     END IF
*
*     RETURN
*
*     End of LSAME.
*
      END
```



```
      SUBROUTINE XERBLA ( SRNAME, INFO )
*     .. Scalar Arguments ..
      INTEGER            INFO
      CHARACTER*6        SRNAME
*     ..
*
*  Purpose
*  =======
*
*  XERBLA  is an error handler for the Level 2 BLAS routines.
*
*  It is called by the Level 2 BLAS routines if an input parameter is
*  invalid.
*
*  Installers should consider modifying the STOP statement in order to
*  call system-specific exception-handling facilities.
*
*  Parameters
*  ==========
*
*  SRNAME - CHARACTER*6.
*           On entry, SRNAME specifies the name of the routine which
*           called XERBLA.
*
*  INFO   - INTEGER.
*           On entry, INFO specifies the position of the invalid
*           parameter in the parameter-list of the calling routine.
*
*
*  Auxiliary routine for Level 2 Blas.
*
*  Written on 20-July-1986.
*
*     .. Executable Statements ..
*
      WRITE (*,99999) SRNAME, INFO
*
      STOP
*
99999 FORMAT ( ' ** On entry to ', A6, ' parameter number ', I2,
     $         ' had an illegal value' )
*
*     End of XERBLA.
*
      END
```

```
      subroutine zaxpy(n,za,zx,incx,zy,incy)
c
c     constant times a vector plus a vector.
c     jack dongarra, 3/11/78.
c
      double complex zx(1),zy(1),za
      double precision dcabs1
      integer n,incx,incy,ix,iy,i
      if(n.le.0)return
      if (dcabs1(za) .eq. 0.0d0) return
      if (incx.eq.1.and.incy.eq.1)go to 20
c
c        code for unequal increments or equal increments
c          not equal to 1
c
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        zy(iy) = zy(iy) + za*zx(ix)
        ix = ix + incx
        iy = iy + incy
   10 continue
      return
c
c        code for both increments equal to 1
c
   20 do 30 i = 1,n
        zy(i) = zy(i) + za*zx(i)
   30 continue
      return
      end
```



```
      double complex function zdotc(n,zx,incx,zy,incy)
c
c     forms the dot product of a vector.
c     jack dongarra, 3/11/78.
c
      double complex zx(1),zy(1),ztemp
      integer n,incx,incy,ix,iy,i
      ztemp = (0.0d0,0.0d0)
      zdotc = (0.0d0,0.0d0)
      if(n.le.0)return
      if(incx.eq.1.and.incy.eq.1)go to 20
c
c        code for unequal increments or equal increments
c          not equal to 1
c
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        ztemp = ztemp + dconjg(zx(ix))*zy(iy)
        ix = ix + incx
        iy = iy + incy
   10 continue
      zdotc = ztemp
      return
c
c        code for both increments equal to 1
c
   20 do 30 i = 1,n
        ztemp = ztemp + dconjg(zx(i))*zy(i)
   30 continue
      zdotc = ztemp
      return
      end
```

```
      subroutine  zdscal(n,da,zx,incx)
c
c     scales a vector by a constant.
c    jack dongarra, 3/11/78.
c
      double complex zx(1)
      double precision da
      integer n,incx,ix,i
      if(n.le.0)return
      if(incx.eq.1)go to 20
c
c        code for increments not equal to 1
c
      ix = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      do 10 i = 1,n
        zx(ix) = dcmplx(da,0.0d0)*zx(ix)
        ix = ix + incx
   10 continue
      return
c
c        code for increments equal to 1
c
   20 do 30 i = 1,n
        zx(i) = dcmplx(da,0.0d0)*zx(i)
   30 continue
      return
      end
```



```
      subroutine zgeco(a,lda,n,ipvt,rcond,z)
      integer lda,n,ipvt(1)
      complex*16 a(lda,1),z(1)
      double precision rcond
c
c     zgeco factors a complex*16 matrix by gaussian elimination
c     and estimates the condition of the matrix.
c
c     if  rcond  is not needed, zgefa is slightly faster.
c     to solve  a*x = b , follow zgeco by zgesl.
c     to compute  inverse(a)*c , follow zgeco by zgesl.
c     to compute  determinant(a) , follow zgeco by zgedi.
c     to compute  inverse(a) , follow zgeco by zgedi.
c
c     on entry
c
c        a       complex*16(lda, n)
c                the matrix to be factored.
c
c        lda     integer
c                the leading dimension of the array  a .
c
c        n       integer
c                the order of the matrix  a .
c
c     on return
c
c        a       an upper triangular matrix and the multipliers
c                which were used to obtain it.
c                the factorization can be written  a = l*u  where
c                l  is a product of permutation and unit lower
c                triangular matrices and  u  is upper triangular.
c
c        ipvt    integer(n)
c                an integer vector of pivot indices.
c
c        rcond   double precision
c                an estimate of the reciprocal condition of  a .
c                for the system  a*x = b , relative perturbations
c                in  a  and  b  of size  epsilon  may cause
c                relative perturbations in  x  of size  epsilon/rcond .
c                if  rcond  is so small that the logical expression
c                           1.0 + rcond .eq. 1.0
c                is true, then  a  may be singular to working
c                precision.  in particular,  rcond  is zero  if
c                exact singularity is detected or the estimate
c                underflows.
c
c        z       complex*16(n)
c                a work vector whose contents are usually unimportant.
c                if  a  is close to a singular matrix, then  z  is
c                an approximate null vector in the sense that
c                norm(a*z) = rcond*norm(a)*norm(z) .
c
c     linpack. this version dated 08/14/78 .
c     cleve moler, university of new mexico, argonne national lab.
c
c     subroutines and functions
c
c     linpack zgefa
c     blas zaxpy,zdotc,zdscal,dzasum
c     fortran dabs,dmax1,dcmplx,dconjg
c
c     internal variables
c
      complex*16 zdotc,ek,t,wk,wkm
      double precision anorm,s,dzasum,sm,ynorm
      integer info,j,k,kb,kp1,l
c
      complex*16 zdum,zdum1,zdum2,csign1
```

```
      double precision cabs1
      double precision dreal,dimag
      complex*16 zdumr,zdumi
      dreal(zdumr) = zdumr
      dimag(zdumi) = (0.0d0,-1.0d0)*zdumi
      cabs1(zdum) = dabs(dreal(zdum)) + dabs(dimag(zdum))
      csign1(zdum1,zdum2) = cabs1(zdum1)*(zdum2/cabs1(zdum2))
c
c     compute 1-norm of a
c
      anorm = 0.0d0
      do 10 j = 1, n
         anorm = dmax1(anorm,dzasum(n,a(1,j),1))
   10 continue
c
c     factor
c
      call zgefa(a,lda,n,ipvt,info)
c
c     rcond = 1/(norm(a)*(estimate of norm(inverse(a)))) .
c     estimate = norm(z)/norm(y) where  a*z = y  and  ctrans(a)*y = e .
c     ctrans(a)  is the conjugate transpose of a .
c     the components of  e  are chosen to cause maximum local
c     growth in the elements of w  where  ctrans(u)*w = e .
c     the vectors are frequently rescaled to avoid overflow.
c
c     solve ctrans(u)*w = e
c
      ek = (1.0d0,0.0d0)
      do 20 j = 1, n
         z(j) = (0.0d0,0.0d0)
   20 continue
      do 100 k = 1, n
         if (cabs1(z(k)) .ne. 0.0d0) ek = csign1(ek,-z(k))
         if (cabs1(ek-z(k)) .le. cabs1(a(k,k))) go to 30
            s = cabs1(a(k,k))/cabs1(ek-z(k))
            call zdscal(n,s,z,1)
            ek = dcmplx(s,0.0d0)*ek
   30    continue
         wk = ek - z(k)
         wkm = -ek - z(k)
         s = cabs1(wk)
         sm = cabs1(wkm)
         if (cabs1(a(k,k)) .eq. 0.0d0) go to 40
            wk = wk/dconjg(a(k,k))
            wkm = wkm/dconjg(a(k,k))
         go to 50
   40    continue
            wk = (1.0d0,0.0d0)
            wkm = (1.0d0,0.0d0)
   50    continue
         kp1 = k + 1
         if (kp1 .gt. n) go to 90
            do 60 j = kp1, n
               sm = sm + cabs1(z(j)+wkm*dconjg(a(k,j)))
               z(j) = z(j) + wk*dconjg(a(k,j))
               s = s + cabs1(z(j))
   60       continue
            if (s .ge. sm) go to 80
               t = wkm - wk
               wk = wkm
               do 70 j = kp1, n
                  z(j) = z(j) + t*dconjg(a(k,j))
   70          continue
   80       continue
   90    continue
         z(k) = wk
  100 continue
      s = 1.0d0/dzasum(n,z,1)
      call zdscal(n,s,z,1)
c
```



```
c     solve ctrans(l)*y = w
c
      do 120 kb = 1, n
         k = n + 1 - kb
         if (k .lt. n) z(k) = z(k) + zdotc(n-k,a(k+1,k),1,z(k+1),1)
         if (cabs1(z(k)) .le. 1.0d0) go to 110
            s = 1.0d0/cabs1(z(k))
            call zdscal(n,s,z,1)
  110    continue
         l = ipvt(k)
         t = z(l)
         z(l) = z(k)
         z(k) = t
  120 continue
      s = 1.0d0/dzasum(n,z,1)
      call zdscal(n,s,z,1)
c
      ynorm = 1.0d0
c
c     solve l*v = y
c
      do 140 k = 1, n
         l = ipvt(k)
         t = z(l)
         z(l) = z(k)
         z(k) = t
         if (k .lt. n) call zaxpy(n-k,t,a(k+1,k),1,z(k+1),1)
         if (cabs1(z(k)) .le. 1.0d0) go to 130
            s = 1.0d0/cabs1(z(k))
            call zdscal(n,s,z,1)
            ynorm = s*ynorm
  130    continue
  140 continue
      s = 1.0d0/dzasum(n,z,1)
      call zdscal(n,s,z,1)
      ynorm = s*ynorm
c
c     solve  u*z = v
c
      do 160 kb = 1, n
         k = n + 1 - kb
         if (cabs1(z(k)) .le. cabs1(a(k,k))) go to 150
            s = cabs1(a(k,k))/cabs1(z(k))
            call zdscal(n,s,z,1)
            ynorm = s*ynorm
  150    continue
         if (cabs1(a(k,k)) .ne. 0.0d0) z(k) = z(k)/a(k,k)
         if (cabs1(a(k,k)) .eq. 0.0d0) z(k) = (1.0d0,0.0d0)
         t = -z(k)
         call zaxpy(k-1,t,a(1,k),1,z(1),1)
  160 continue
c     make znorm = 1.0
      s = 1.0d0/dzasum(n,z,1)
      call zdscal(n,s,z,1)
      ynorm = s*ynorm
c
      if (anorm .ne. 0.0d0) rcond = ynorm/anorm
      if (anorm .eq. 0.0d0) rcond = 0.0d0
      return
      end
```

```
      subroutine zgefa(a,lda,n,ipvt,info)
      integer lda,n,ipvt(1),info
      complex*16 a(lda,1)
c
c     zgefa factors a complex*16 matrix by gaussian elimination.
c
c     zgefa is usually called by zgeco, but it can be called
c     directly with a saving in time if  rcond  is not needed.
c     (time for zgeco) = (1 + 9/n)*(time for zgefa) .
c
c     on entry
c
c        a       complex*16(lda, n)
c                the matrix to be factored.
c
c        lda     integer
c                the leading dimension of the array  a .
c
c        n       integer
c                the order of the matrix  a .
c
c     on return
c
c        a       an upper triangular matrix and the multipliers
c                which were used to obtain it.
c                the factorization can be written  a = l*u  where
c                l  is a product of permutation and unit lower
c                triangular matrices and  u  is upper triangular.
c
c        ipvt    integer(n)
c                an integer vector of pivot indices.
c
c        info    integer
c                = 0  normal value.
c                = k  if  u(k,k) .eq. 0.0 .  this is not an error
c                     condition for this subroutine, but it does
c                     indicate that zgesl or zgedi will divide by zero
c                     if called.  use  rcond  in zgeco for a reliable
c                     indication of singularity.
c
c     linpack. this version dated 08/14/78 .
c     cleve moler, university of new mexico, argonne national lab.
c
c     subroutines and functions
c
c     blas zaxpy,zscal,izamax
c     fortran dabs
c
c     internal variables
c
      complex*16 t
      integer izamax,j,k,kp1,l,nm1
c
      complex*16 zdum
      double precision cabs1
      double precision dreal,dimag
      complex*16 zdumr,zdumi
      dreal(zdumr) = zdumr
      dimag(zdumi) = (0.0d0,-1.0d0)*zdumi
      cabs1(zdum) = dabs(dreal(zdum)) + dabs(dimag(zdum))
c
c     gaussian elimination with partial pivoting
c
      info = 0
      nm1 = n - 1
      if (nm1 .lt. 1) go to 70
      do 60 k = 1, nm1
         kp1 = k + 1
c
c        find l = pivot index
```



```
c
         l = izamax(n-k+1,a(k,k),1) + k - 1
         ipvt(k) = l
c
c        zero pivot implies this column already triangularized
c
         if (cabs1(a(l,k)) .eq. 0.0d0) go to 40
c
c           interchange if necessary
c
            if (l .eq. k) go to 10
               t = a(l,k)
               a(l,k) = a(k,k)
               a(k,k) = t
   10       continue
c
c           compute multipliers
c
            t = -(1.0d0,0.0d0)/a(k,k)
            call zscal(n-k,t,a(k+1,k),1)
c
c           row elimination with column indexing
c
            do 30 j = kp1, n
               t = a(l,j)
               if (l .eq. k) go to 20
                  a(l,j) = a(k,j)
                  a(k,j) = t
   20          continue
               call zaxpy(n-k,t,a(k+1,k),1,a(k+1,j),1)
   30       continue
         go to 50
   40    continue
            info = k
   50    continue
   60 continue
   70 continue
      ipvt(n) = n
      if (cabs1(a(n,n)) .eq. 0.0d0) info = n
      return
      end
```

```
*
*  -- Written on 8-February-1989.
*     Jack Dongarra, Argonne National Laboratory.
*     Iain Duff, AERE Harwell.
*     Jeremy Du Croz, Numerical Algorithms Group Ltd.
*     Sven Hammarling, Numerical Algorithms Group Ltd.
*
*  =====================================================================
*
*     .. External Functions ..
      LOGICAL            LSAME
      EXTERNAL           LSAME
*     .. External Subroutines ..
      EXTERNAL           XERBLA
*     .. Intrinsic Functions ..
      INTRINSIC          DCONJG, MAX
*     .. Scalar Arguments ..
      CHARACTER*1        TRANSA, TRANSB
      INTEGER            M, N, K, LDA, LDB, LDC
      COMPLEX*16         ALPHA, BETA
*     .. Array Arguments ..
      COMPLEX*16         A( LDA, * ), B( LDB, * ), C( LDC, * )
*     ..
*
*     Set  NOTA  and  NOTB  as  true if  A  and  B  respectively are not
*     conjugated or transposed, set  CONJA and CONJB  as true if  A  and
*     B  respectively are to be  transposed but  not conjugated  and set
*     NROWA, NCOLA and  NROWB  as the number of rows and  columns  of  A
*     and the number of rows of  B  respectively.
*
      NOTA  = LSAME( TRANSA, 'N' )
      NOTB  = LSAME( TRANSB, 'N' )
      CONJA = LSAME( TRANSA, 'C' )
      CONJB = LSAME( TRANSB, 'C' )
      IF( NOTA )THEN
         NROWA = M
         NCOLA = K
      ELSE
         NROWA = K
         NCOLA = M
      END IF
      IF( NOTB )THEN
         NROWB = K
      ELSE
         NROWB = N
      END IF
*
*     Test the input parameters.
*
      INFO = 0
      IF(      ( .NOT.NOTA  ).AND.
     $         ( .NOT.CONJA ).AND.
     $         ( .NOT.LSAME( TRANSA, 'T' ) ).AND.
     $         ( .NOT.LSAME( TRANSA, 't' ) )      )THEN
         INFO = 1
      ELSE IF( ( .NOT.NOTB  ).AND.
     $         ( .NOT.CONJB ).AND.
     $         ( .NOT.LSAME( TRANSB, 'T' ) ).AND.
     $         ( .NOT.LSAME( TRANSB, 't' ) )      )THEN
         INFO = 2
      ELSE IF( M.LT.0 )THEN
         INFO = 3
      ELSE IF( N.LT.0 )THEN
         INFO = 4
      ELSE IF( K.LT.0 )THEN
         INFO = 5
      ELSE IF( LDA.LT.MAX( 1, NROWA ) )THEN
         INFO = 8
      ELSE IF( LDB.LT.MAX( 1, NROWB ) )THEN
         INFO = 10
      ELSE IF( LDC.LT.MAX( 1, M ) )THEN
         INFO = 13
      END IF
      IF( INFO.NE.0 )THEN
         CALL XERBLA( 'ZGEMM ', INFO )
         RETURN
      END IF
*
*     Quick return if possible.
*
      IF( ( M.EQ.0 ).OR.( N.EQ.0 ).OR.
     $    ( ( ( ALPHA.EQ.ZERO ).OR.( K.EQ.0 ) ).AND.( BETA.EQ.ONE ) ) )
     $   RETURN
*
*     And when  alpha.eq.zero.
*
      IF( ALPHA.EQ.ZERO )THEN
         IF( BETA.EQ.ZERO )THEN
            DO 20, J = 1, N
               DO 10, I = 1, M
                  C( I, J ) = ZERO
   10          CONTINUE
   20       CONTINUE
         ELSE
            DO 40, J = 1, N
               DO 30, I = 1, M
                  C( I, J ) = BETA*C( I, J )
   30          CONTINUE
   40       CONTINUE
         END IF
         RETURN
      END IF
*
*     Start the operations.
*
      IF( NOTB )THEN
         IF( NOTA )THEN
*
*           Form  C := alpha*A*B + beta*C.
*
            DO 90, J = 1, N
               IF( BETA.EQ.ZERO )THEN
                  DO 50, I = 1, M
                     C( I, J ) = ZERO
   50             CONTINUE
               ELSE IF( BETA.NE.ONE )THEN
                  DO 60, I = 1, M
                     C( I, J ) = BETA*C( I, J )
   60             CONTINUE
               END IF
               DO 80, L = 1, K
                  IF( B( L, J ).NE.ZERO )THEN
                     TEMP = ALPHA*B( L, J )
                     DO 70, I = 1, M
                        C( I, J ) = C( I, J ) + TEMP*A( I, L )
   70                CONTINUE
                  END IF
   80          CONTINUE
   90       CONTINUE
         ELSE IF( CONJA )THEN
*
*           Form  C := alpha*conjg( A' )*B + beta*C.
*
            DO 120, J = 1, N
               DO 110, I = 1, M
                  TEMP = ZERO
                  DO 100, L = 1, K
                     TEMP = TEMP + DCONJG( A( L, I ) )*B( L, J )
  100             CONTINUE
                  IF( BETA.EQ.ZERO )THEN
                     C( I, J ) = ALPHA*TEMP
                  ELSE
                     C( I, J ) = ALPHA*TEMP + BETA*C( I, J )
                  END IF
  110          CONTINUE
  120       CONTINUE
         ELSE
*
*           Form  C := alpha*A'*B + beta*C
*
            DO 150, J = 1, N
               DO 140, I = 1, M
                  TEMP = ZERO
                  DO 130, L = 1, K
                     TEMP = TEMP + A( L, I )*B( L, J )
  130             CONTINUE
                  IF( BETA.EQ.ZERO )THEN
                     C( I, J ) = ALPHA*TEMP
                  ELSE
                     C( I, J ) = ALPHA*TEMP + BETA*C( I, J )
                  END IF
  140          CONTINUE
  150       CONTINUE
         END IF
      ELSE IF( NOTA )THEN
         IF( CONJB )THEN
*
*           Form  C := alpha*A*conjg( B' ) + beta*C.
*
            DO 200, J = 1, N
               IF( BETA.EQ.ZERO )THEN
                  DO 160, I = 1, M
                     C( I, J ) = ZERO
  160             CONTINUE
               ELSE IF( BETA.NE.ONE )THEN
                  DO 170, I = 1, M
                     C( I, J ) = BETA*C( I, J )
  170             CONTINUE
               END IF
               DO 190, L = 1, K
                  IF( B( J, L ).NE.ZERO )THEN
                     TEMP = ALPHA*DCONJG( B( J, L ) )
                     DO 180, I = 1, M
                        C( I, J ) = C( I, J ) + TEMP*A( I, L )
  180                CONTINUE
                  END IF
  190          CONTINUE
  200       CONTINUE
         ELSE
*
*           Form  C := alpha*A*B' + beta*C
*
            DO 250, J = 1, N
               IF( BETA.EQ.ZERO )THEN
                  DO 210, I = 1, M
                     C( I, J ) = ZERO
  210             CONTINUE
               ELSE IF( BETA.NE.ONE )THEN
                  DO 220, I = 1, M
                     C( I, J ) = BETA*C( I, J )
  220             CONTINUE
               END IF
               DO 240, L = 1, K
                  IF( B( J, L ).NE.ZERO )THEN
                     TEMP = ALPHA*B( J, L )
                     DO 230, I = 1, M
                        C( I, J ) = C( I, J ) + TEMP*A( I, L )
  230                CONTINUE
                  END IF
  240          CONTINUE
  250       CONTINUE
         END IF
      ELSE IF( CONJA )THEN
         IF( CONJB )THEN
*
*           Form  C := alpha*conjg( A' )*conjg( B' ) + beta*C.
*
            DO 280, J = 1, N
               DO 270, I = 1, M
                  TEMP = ZERO
                  DO 260, L = 1, K
                     TEMP = TEMP + DCONJG( A( L, I ) )*DCONJG( B( J, L ) )
  260             CONTINUE
                  IF( BETA.EQ.ZERO )THEN
                     C( I, J ) = ALPHA*TEMP
                  ELSE
                     C( I, J ) = ALPHA*TEMP + BETA*C( I, J )
                  END IF
  270          CONTINUE
  280       CONTINUE
         ELSE
*
*           Form  C := alpha*conjg( A' )*B' + beta*C
*
            DO 310, J = 1, N
               DO 300, I = 1, M
                  TEMP = ZERO
                  DO 290, L = 1, K
                     TEMP = TEMP + DCONJG( A( L, I ) )*B( J, L )
  290             CONTINUE
                  IF( BETA.EQ.ZERO )THEN
                     C( I, J ) = ALPHA*TEMP
                  ELSE
                     C( I, J ) = ALPHA*TEMP + BETA*C( I, J )
                  END IF
  300          CONTINUE
  310       CONTINUE
         END IF
      ELSE
         IF( CONJB )THEN
*
*           Form  C := alpha*A'*conjg( B' ) + beta*C
*
            DO 340, J = 1, N
               DO 330, I = 1, M
                  TEMP = ZERO
                  DO 320, L = 1, K
                     TEMP = TEMP + A( L, I )*DCONJG( B( J, L ) )
  320             CONTINUE
                  IF( BETA.EQ.ZERO )THEN
                     C( I, J ) = ALPHA*TEMP
                  ELSE
                     C( I, J ) = ALPHA*TEMP + BETA*C( I, J )
                  END IF
  330          CONTINUE
  340       CONTINUE
         ELSE
*
*           Form  C := alpha*A'*B' + beta*C
*
            DO 370, J = 1, N
               DO 360, I = 1, M
                  TEMP = ZERO
                  DO 350, L = 1, K
                     TEMP = TEMP + A( L, I )*B( J, L )
  350             CONTINUE
                  IF( BETA.EQ.ZERO )THEN
                     C( I, J ) = ALPHA*TEMP
                  ELSE
                     C( I, J ) = ALPHA*TEMP + BETA*C( I, J )
                  END IF
  360          CONTINUE
  370       CONTINUE
         END IF
      END IF
*
      RETURN
*
*     End of ZGEMM .
*
      END
```

```
*     .. Local Scalars ..
      LOGICAL            CONJA, CONJB, NOTA, NOTB
      INTEGER            I, INFO, J, L, NCOLA, NROWA, NROWB
      COMPLEX*16         TEMP
*     .. Parameters ..
      COMPLEX*16         ONE
      PARAMETER        ( ONE  = ( 1.0D+0, 0.0D+0 ) )
      COMPLEX*16         ZERO
      PARAMETER        ( ZERO = ( 0.0D+0, 0.0D+0 ) )
*     ..
*     .. Executable Statements ..
*
*     Set  NOTA  and  NOTB  as  true if  A  and  B  respectively are not
*     conjugated or transposed, set  CONJA and CONJB  as true if  A  and
*     B  respectively are to be  transposed but  not conjugated  and set
*     NROWA, NCOLA and  NROWB  as the number of rows and  columns  of  A
*     and the number of rows of  B  respectively.
*
      NOTA  = LSAME( TRANSA, 'N' )
      NOTB  = LSAME( TRANSB, 'N' )
      CONJA = LSAME( TRANSA, 'C' )
      CONJB = LSAME( TRANSB, 'C' )
      IF( NOTA )THEN
         NROWA = M
         NCOLA = K
      ELSE
         NROWA = K
         NCOLA = M
      END IF
      IF( NOTB )THEN
         NROWB = K
      ELSE
         NROWB = N
      END IF
*
*     Test the input parameters.
*
      INFO = 0
      IF(      ( .NOT.NOTA                 ).AND.
     $         ( .NOT.CONJA                ).AND.
     $         ( .NOT.LSAME( TRANSA, 'T' ) )      )THEN
         INFO = 1
      ELSE IF( ( .NOT.NOTB                 ).AND.
     $         ( .NOT.CONJB                ).AND.
     $         ( .NOT.LSAME( TRANSB, 'T' ) )      )THEN
         INFO = 2
      ELSE IF( M  .LT.0                )THEN
         INFO = 3
      ELSE IF( N  .LT.0                )THEN
         INFO = 4
      ELSE IF( K  .LT.0                )THEN
         INFO = 5
      ELSE IF( LDA.LT.MAX( 1, NROWA ) )THEN
         INFO = 8
      ELSE IF( LDB.LT.MAX( 1, NROWB ) )THEN
         INFO = 10
      ELSE IF( LDC.LT.MAX( 1, M     ) )THEN
         INFO = 13
      END IF
      IF( INFO.NE.0 )THEN
         CALL XERBLA( 'ZGEMM ', INFO )
         RETURN
      END IF
*
*     Quick return if possible.
*
      IF( ( M.EQ.0 ).OR.( N.EQ.0 ).OR.
     $    ( ( ( ALPHA.EQ.ZERO ).OR.( K.EQ.0 ) ).AND.( BETA.EQ.ONE ) ) )
     $   RETURN
*
*     And when  alpha.eq.zero.
```



```
*
      IF( ALPHA.EQ.ZERO )THEN
         IF( BETA.EQ.ZERO )THEN
            DO 20, J = 1, N
               DO 10, I = 1, M
                  C( I, J ) = ZERO
   10          CONTINUE
   20       CONTINUE
         ELSE
            DO 40, J = 1, N
               DO 30, I = 1, M
                  C( I, J ) = BETA*C( I, J )
   30          CONTINUE
   40       CONTINUE
         END IF
         RETURN
      END IF
*
*     Start the operations.
*
      IF( NOTB )THEN
         IF( NOTA )THEN
*
*           Form  C := alpha*A*B + beta*C.
*
            DO 90, J = 1, N
               IF( BETA.EQ.ZERO )THEN
                  DO 50, I = 1, M
                     C( I, J ) = ZERO
   50             CONTINUE
               ELSE IF( BETA.NE.ONE )THEN
                  DO 60, I = 1, M
                     C( I, J ) = BETA*C( I, J )
   60             CONTINUE
               END IF
               DO 80, L = 1, K
                  IF( B( L, J ).NE.ZERO )THEN
                     TEMP = ALPHA*B( L, J )
                     DO 70, I = 1, M
                        C( I, J ) = C( I, J ) + TEMP*A( I, L )
   70                CONTINUE
                  END IF
   80          CONTINUE
   90       CONTINUE
         ELSE IF( CONJA )THEN
*
*           Form  C := alpha*conjg( A' )*B + beta*C.
*
            DO 120, J = 1, N
               DO 110, I = 1, M
                  TEMP = ZERO
                  DO 100, L = 1, K
                     TEMP = TEMP + DCONJG( A( L, I ) )*B( L, J )
  100             CONTINUE
                  IF( BETA.EQ.ZERO )THEN
                     C( I, J ) = ALPHA*TEMP
                  ELSE
                     C( I, J ) = ALPHA*TEMP + BETA*C( I, J )
                  END IF
  110          CONTINUE
  120       CONTINUE
         ELSE
*
*           Form  C := alpha*A'*B + beta*C
*
            DO 150, J = 1, N
               DO 140, I = 1, M
                  TEMP = ZERO
                  DO 130, L = 1, K
                     TEMP = TEMP + A( L, I )*B( L, J )
  130             CONTINUE
```

```
                  IF( BETA.EQ.ZERO )THEN
                     C( I, J ) = ALPHA*TEMP
                  ELSE
                     C( I, J ) = ALPHA*TEMP + BETA*C( I, J )
                  END IF
  140          CONTINUE
  150       CONTINUE
         END IF
      ELSE IF( NOTA )THEN
         IF( CONJB )THEN
*
*           Form  C := alpha*A*conjg( B' ) + beta*C.
*
            DO 200, J = 1, N
               IF( BETA.EQ.ZERO )THEN
                  DO 160, I = 1, M
                     C( I, J ) = ZERO
  160             CONTINUE
               ELSE IF( BETA.NE.ONE )THEN
                  DO 170, I = 1, M
                     C( I, J ) = BETA*C( I, J )
  170             CONTINUE
               END IF
               DO 190, L = 1, K
                  IF( B( J, L ).NE.ZERO )THEN
                     TEMP = ALPHA*DCONJG( B( J, L ) )
                     DO 180, I = 1, M
                        C( I, J ) = C( I, J ) + TEMP*A( I, L )
  180                CONTINUE
                  END IF
  190          CONTINUE
  200       CONTINUE
         ELSE
*
*           Form  C := alpha*A*B'          + beta*C
*
            DO 250, J = 1, N
               IF( BETA.EQ.ZERO )THEN
                  DO 210, I = 1, M
                     C( I, J ) = ZERO
  210             CONTINUE
               ELSE IF( BETA.NE.ONE )THEN
                  DO 220, I = 1, M
                     C( I, J ) = BETA*C( I, J )
  220             CONTINUE
               END IF
               DO 240, L = 1, K
                  IF( B( J, L ).NE.ZERO )THEN
                     TEMP = ALPHA*B( J, L )
                     DO 230, I = 1, M
                        C( I, J ) = C( I, J ) + TEMP*A( I, L )
  230                CONTINUE
                  END IF
  240          CONTINUE
  250       CONTINUE
         END IF
      ELSE IF( CONJA )THEN
         IF( CONJB )THEN
*
*           Form  C := alpha*conjg( A' )*conjg( B' ) + beta*C.
*
            DO 280, J = 1, N
               DO 270, I = 1, M
                  TEMP = ZERO
                  DO 260, L = 1, K
                     TEMP = TEMP +
     $                    DCONJG( A( L, I ) )*DCONJG( B( J, L ) )
  260             CONTINUE
                  IF( BETA.EQ.ZERO )THEN
                     C( I, J ) = ALPHA*TEMP
                  ELSE
```



```
                     C( I, J ) = ALPHA*TEMP + BETA*C( I, J )
                  END IF
  270          CONTINUE
  280       CONTINUE
         ELSE
*
*           Form  C := alpha*conjg( A' )*B' + beta*C
*
            DO 310, J = 1, N
               DO 300, I = 1, M
                  TEMP = ZERO
                  DO 290, L = 1, K
                     TEMP = TEMP + DCONJG( A( L, I ) )*B( J, L )
  290             CONTINUE
                  IF( BETA.EQ.ZERO )THEN
                     C( I, J ) = ALPHA*TEMP
                  ELSE
                     C( I, J ) = ALPHA*TEMP + BETA*C( I, J )
                  END IF
  300          CONTINUE
  310       CONTINUE
         END IF
      ELSE
         IF( CONJB )THEN
*
*           Form  C := alpha*A'*conjg( B' ) + beta*C
*
            DO 340, J = 1, N
               DO 330, I = 1, M
                  TEMP = ZERO
                  DO 320, L = 1, K
                     TEMP = TEMP + A( L, I )*DCONJG( B( J, L ) )
  320             CONTINUE
                  IF( BETA.EQ.ZERO )THEN
                     C( I, J ) = ALPHA*TEMP
                  ELSE
                     C( I, J ) = ALPHA*TEMP + BETA*C( I, J )
                  END IF
  330          CONTINUE
  340       CONTINUE
         ELSE
*
*           Form  C := alpha*A'*B' + beta*C
*
            DO 370, J = 1, N
               DO 360, I = 1, M
                  TEMP = ZERO
                  DO 350, L = 1, K
                     TEMP = TEMP + A( L, I )*B( J, L )
  350             CONTINUE
                  IF( BETA.EQ.ZERO )THEN
                     C( I, J ) = ALPHA*TEMP
                  ELSE
                     C( I, J ) = ALPHA*TEMP + BETA*C( I, J )
                  END IF
  360          CONTINUE
  370       CONTINUE
         END IF
      END IF
*
      RETURN
*
*     End of ZGEMM .
*
      END
```

```
C This file contains routines for matrix-vector manipulations,
C routines for solution of linear equations with complex matrix,
C the routines inprod and EorF for integration and the function
C fase.
C 23 march 1992.
*
*  =====================================================================
*
      SUBROUTINE ZGEMV ( TRANS, M, N, ALPHA, A, LDA, X, INCX,
     $                   BETA, Y, INCY )
*     .. Scalar Arguments ..
      COMPLEX*16         ALPHA, BETA
      INTEGER            INCX, INCY, LDA, M, N
      CHARACTER*1        TRANS
*     .. Array Arguments ..
      COMPLEX*16         A( LDA, * ), X( * ), Y( * )
*     ..
*
*  Purpose
*  =======
*
*  ZGEMV  performs one of the matrix-vector operations
*
*     y := alpha*A*x + beta*y,   or   y := alpha*A'*x + beta*y,   or
*
*     y := alpha*conjg( A' )*x + beta*y,
*
*  where alpha and beta are scalars, x and y are vectors and A is an
*  m by n matrix.
*
*  Parameters
*  ==========
*
*  TRANS  - CHARACTER*1.
*           On entry, TRANS specifies the operation to be performed as
*           follows:
*
*              TRANS = 'N' or 'n'   y := alpha*A*x + beta*y.
*
*              TRANS = 'T' or 't'   y := alpha*A'*x + beta*y.
*
*              TRANS = 'C' or 'c'   y := alpha*conjg( A' )*x + beta*y.
*
*           Unchanged on exit.
*
*  M      - INTEGER.
*           On entry, M specifies the number of rows of the matrix A.
*           M must be at least zero.
*           Unchanged on exit.
*
*  N      - INTEGER.
*           On entry, N specifies the number of columns of the matrix A.
*           N must be at least zero.
*           Unchanged on exit.
*
*  ALPHA  - COMPLEX*16      .
*           On entry, ALPHA specifies the scalar alpha.
*           Unchanged on exit.
*
*  A      - COMPLEX*16       array of DIMENSION ( LDA, n ).
*           Before entry, the leading m by n part of the array A must
*           contain the matrix of coefficients.
*           Unchanged on exit.
*
*  LDA    - INTEGER.
*           On entry, LDA specifies the first dimension of A as declared
*           in the calling (sub) program. LDA must be at least
*           max( 1, m ).
*           Unchanged on exit.
*
```



```
*  X      - COMPLEX*16       array of DIMENSION at least
*           ( 1 + ( n - 1 )*abs( INCX ) ) when TRANS = 'N' or 'n'
*           and at least
*           ( 1 + ( m - 1 )*abs( INCX ) ) otherwise.
*           Before entry, the incremented array X must contain the
*           vector x.
*           Unchanged on exit.
*
*  INCX   - INTEGER.
*           On entry, INCX specifies the increment for the elements of
*           X. INCX must not be zero.
*           Unchanged on exit.
*
*  BETA   - COMPLEX*16      .
*           On entry, BETA specifies the scalar beta. When BETA is
*           supplied as zero then Y need not be set on input.
*           Unchanged on exit.
*
*  Y      - COMPLEX*16       array of DIMENSION at least
*           ( 1 + ( m - 1 )*abs( INCY ) ) when TRANS = 'N' or 'n'
*           and at least
*           ( 1 + ( n - 1 )*abs( INCY ) ) otherwise.
*           Before entry with BETA non-zero, the incremented array Y
*           must contain the vector y. On exit, Y is overwritten by the
*           updated vector y.
*
*  INCY   - INTEGER.
*           On entry, INCY specifies the increment for the elements of
*           Y. INCY must not be zero.
*           Unchanged on exit.
*
*
*  Level 2 Blas routine.
*
*  -- Written on 22-October-1986.
*     Jack Dongarra, Argonne National Lab.
*     Jeremy Du Croz, Nag Central Office.
*     Sven Hammarling, Nag Central Office.
*     Richard Hanson, Sandia National Labs.
*
*
*     .. Parameters ..
      COMPLEX*16         ONE
      PARAMETER        ( ONE  = ( 1.0D+0, 0.0D+0 ) )
      COMPLEX*16         ZERO
      PARAMETER        ( ZERO = ( 0.0D+0, 0.0D+0 ) )
*     .. Local Scalars ..
      COMPLEX*16         TEMP
      INTEGER            I, INFO, IX, IY, J, JX, JY, KX, KY, LENX, LENY
      LOGICAL            NOCONJ
*     .. External Functions ..
      LOGICAL            LSAME
      EXTERNAL           LSAME
*     .. External Subroutines ..
      EXTERNAL           XERBLA
*     .. Intrinsic Functions ..
      INTRINSIC          DCONJG, MAX
*     ..
*     .. Executable Statements ..
*
*     Test the input parameters.
*
      INFO = 0
      IF     ( .NOT.LSAME( TRANS, 'N' ).AND.
     $         .NOT.LSAME( TRANS, 'T' ).AND.
     $         .NOT.LSAME( TRANS, 'C' )      )THEN
         INFO = 1
      ELSE IF( M.LT.0 )THEN
         INFO = 2
      ELSE IF( N.LT.0 )THEN
         INFO = 3
```

```
      ELSE IF( LDA.LT.MAX( 1, M ) )THEN
         INFO = 6
      ELSE IF( INCX.EQ.0 )THEN
         INFO = 8
      ELSE IF( INCY.EQ.0 )THEN
         INFO = 11
      END IF
      IF( INFO.NE.0 )THEN
         CALL XERBLA( 'ZGEMV ', INFO )
         RETURN
      END IF
*
*     Quick return if possible.
*
      IF( ( M.EQ.0 ).OR.( N.EQ.0 ).OR.
     $    ( ( ALPHA.EQ.ZERO ).AND.( BETA.EQ.ONE ) ) )
     $   RETURN
*
      NOCONJ = LSAME( TRANS, 'T' )
*
*     Set  LENX  and  LENY, the lengths of the vectors x and y, and set
*     up the start points in  X  and  Y.
*
      IF( LSAME( TRANS, 'N' ) )THEN
         LENX = N
         LENY = M
      ELSE
         LENX = M
         LENY = N
      END IF
      IF( INCX.GT.0 )THEN
         KX = 1
      ELSE
         KX = 1 - ( LENX - 1 )*INCX
      END IF
      IF( INCY.GT.0 )THEN
         KY = 1
      ELSE
         KY = 1 - ( LENY - 1 )*INCY
      END IF
*
*     Start the operations. In this version the elements of A are
*     accessed sequentially with one pass through A.
*
*     First form  y := beta*y.
*
      IF( BETA.NE.ONE )THEN
         IF( INCY.EQ.1 )THEN
            IF( BETA.EQ.ZERO )THEN
               DO 10, I = 1, LENY
                  Y( I ) = ZERO
   10          CONTINUE
            ELSE
               DO 20, I = 1, LENY
                  Y( I ) = BETA*Y( I )
   20          CONTINUE
            END IF
         ELSE
            IY = KY
            IF( BETA.EQ.ZERO )THEN
               DO 30, I = 1, LENY
                  Y( IY ) = ZERO
                  IY      = IY   + INCY
   30          CONTINUE
            ELSE
               DO 40, I = 1, LENY
                  Y( IY ) = BETA*Y( IY )
                  IY      = IY           + INCY
   40          CONTINUE
            END IF
         END IF
```



```
      END IF
      IF( ALPHA.EQ.ZERO )
     $   RETURN
      IF( LSAME( TRANS, 'N' ) )THEN
*
*        Form  y := alpha*A*x + y.
*
         JX = KX
         IF( INCY.EQ.1 )THEN
            DO 60, J = 1, N
               IF( X( JX ).NE.ZERO )THEN
                  TEMP = ALPHA*X( JX )
                  DO 50, I = 1, M
                     Y( I ) = Y( I ) + TEMP*A( I, J )
   50             CONTINUE
               END IF
               JX = JX + INCX
   60       CONTINUE
         ELSE
            DO 80, J = 1, N
               IF( X( JX ).NE.ZERO )THEN
                  TEMP = ALPHA*X( JX )
                  IY   = KY
                  DO 70, I = 1, M
                     Y( IY ) = Y( IY ) + TEMP*A( I, J )
                     IY      = IY      + INCY
   70             CONTINUE
               END IF
               JX = JX + INCX
   80       CONTINUE
         END IF
      ELSE
*
*        Form  y := alpha*A'*x + y  or  y := alpha*conjg( A' )*x + y.
*
         JY = KY
         IF( INCX.EQ.1 )THEN
            DO 110, J = 1, N
               TEMP = ZERO
               IF( NOCONJ )THEN
                  DO 90, I = 1, M
                     TEMP = TEMP + A( I, J )*X( I )
   90             CONTINUE
               ELSE
                  DO 100, I = 1, M
                     TEMP = TEMP + DCONJG( A( I, J ) )*X( I )
  100             CONTINUE
               END IF
               Y( JY ) = Y( JY ) + ALPHA*TEMP
               JY      = JY      + INCY
  110       CONTINUE
         ELSE
            DO 140, J = 1, N
               TEMP = ZERO
               IX   = KX
               IF( NOCONJ )THEN
                  DO 120, I = 1, M
                     TEMP = TEMP + A( I, J )*X( IX )
                     IX   = IX   + INCX
  120             CONTINUE
               ELSE
                  DO 130, I = 1, M
                     TEMP = TEMP + DCONJG( A( I, J ) )*X( IX )
                     IX   = IX   + INCX
  130             CONTINUE
               END IF
               Y( JY ) = Y( JY ) + ALPHA*TEMP
               JY      = JY      + INCY
  140       CONTINUE
         END IF
      END IF
```

```
*
      RETURN
*
*     End of ZGEMV .
*
      END
```



```
      subroutine zgesl(a,lda,n,ipvt,b,job)
      integer lda,n,ipvt(1),job
      complex*16 a(lda,1),b(1)
c
c     zgesl solves the complex*16 system
c     a * x = b  or  ctrans(a) * x = b
c     using the factors computed by zgeco or zgefa.
c
c     on entry
c
c        a       complex*16(lda, n)
c                the output from zgeco or zgefa.
c
c        lda     integer
c                the leading dimension of the array  a .
c
c        n       integer
c                the order of the matrix  a .
c
c        ipvt    integer(n)
c                the pivot vector from zgeco or zgefa.
c
c        b       complex*16(n)
c                the right hand side vector.
c
c        job     integer
c                = 0         to solve  a*x = b ,
c                = nonzero   to solve  ctrans(a)*x = b  where
c                            ctrans(a)  is the conjugate transpose.
c
c     on return
c
c        b       the solution vector  x .
c
c     error condition
c
c        a division by zero will occur if the input factor contains a
c        zero on the diagonal.  technically this indicates singularity
c        but it is often caused by improper arguments or improper
c        setting of lda .  it will not occur if the subroutines are
c        called correctly and if zgeco has set rcond .gt. 0.0
c        or zgefa has set info .eq. 0 .
c
c     to compute  inverse(a) * c  where  c  is a matrix
c     with  p  columns
c           call zgeco(a,lda,n,ipvt,rcond,z)
c           if (rcond is too small) go to ...
c           do 10 j = 1, p
c              call zgesl(a,lda,n,ipvt,c(1,j),0)
c        10 continue
c
c     linpack. this version dated 08/14/78 .
c     cleve moler, university of new mexico, argonne national lab.
c
c     subroutines and functions
c
c     blas zaxpy,zdotc
c     fortran dconjg
c
c     internal variables
c
      complex*16 zdotc,t
      integer k,kb,l,nm1
      double precision dreal,dimag
      complex*16 zdumr,zdumi
      dreal(zdumr) = zdumr
      dimag(zdumi) = (0.0d0,-1.0d0)*zdumi
c
      nm1 = n - 1
      if (job .ne. 0) go to 50
```

```
c
c        job = 0 , solve  a * x = b
c        first solve  l*y = b
c
         if (nm1 .lt. 1) go to 30
         do 20 k = 1, nm1
            l = ipvt(k)
            t = b(l)
            if (l .eq. k) go to 10
               b(l) = b(k)
               b(k) = t
   10       continue
            call zaxpy(n-k,t,a(k+1,k),1,b(k+1),1)
   20    continue
   30    continue
c
c        now solve  u*x = y
c
         do 40 kb = 1, n
            k = n + 1 - kb
            b(k) = b(k)/a(k,k)
            t = -b(k)
            call zaxpy(k-1,t,a(1,k),1,b(1),1)
   40    continue
      go to 100
   50 continue
c
c        job = nonzero, solve  ctrans(a) * x = b
c        first solve  ctrans(u)*y = b
c
         do 60 k = 1, n
            t = zdotc(k-1,a(1,k),1,b(1),1)
            b(k) = (b(k) - t)/dconjg(a(k,k))
   60    continue
c
c        now solve ctrans(l)*x = y
c
         if (nm1 .lt. 1) go to 90
         do 80 kb = 1, nm1
            k = n - kb
            b(k) = b(k) + zdotc(n-k,a(k+1,k),1,b(k+1),1)
            l = ipvt(k)
            if (l .eq. k) go to 70
               t = b(l)
               b(l) = b(k)
               b(k) = t
   70       continue
   80    continue
   90    continue
  100 continue
      return
      end
```



```
      subroutine  zscal(n,za,zx,incx)
c
c    scales a vector by a constant.
c    jack dongarra, 3/11/78.
c
      double complex za,zx(1)
      integer n,incx,ix,i
      if(n.le.0)return
      if(incx.eq.1)go to 20
c
c        code for increments not equal to 1
c
      ix = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      do 10 i = 1,n
        zx(ix) = za*zx(ix)
        ix = ix + incx
   10 continue
      return
c
c        code for increments equal to 1
c
   20 do 30 i = 1,n
        zx(i) = za*zx(i)
   30 continue
      return
      end
```