December 7, 2014
0:31

Contents

1 INTRODUCTION

This testbench program is the test for UHF (rather DODS) calculation of H2O, including generation of 1- and 2-electron integrals. The ERI are stored in the file *fort*.17 which is a binary file of size 8K. The working molecule is water in STO3G minimal basis set.

```
  /* STRUCTURES */
"main.f" 1.1 ≡
  @m YES 0
  @m NO 100

  @m ERR −10
  @m OK 10

  @m END_OF_FILE −1
  @m NOT_END_OF_FILE 55

  @m LAST_BLOCK 12
  @m NOT_LAST_BLOCK −12

        /∗ OPERATIONAL CONSTANTS ∗/

  @m ARB 1

  @m BYTES_PER_INTEGER 4
  @m LEAST_BYTE 1

  @m NO_OF_TYPES 20
  @m INT_BLOCK_SIZE 20

  @m MAX_BASIS_FUNCTIONS 255
  @m MAX_PRIMITIVES 1000
  @m MAX_CENTRES 50

  @m MAX_ITERATIONS 60

  @m UHF_CALCULATION 11
  @m CLOSED_SHELL_CALCULATION 21

  @m MATRIX_SIZE 20000

        /∗ OUTPUT STREAM UNITS ∗/

  @m ERROR_OUTPUT_UNIT 6
  @m ERI_UNIT 17
@Ln:
```

2 DODS

[dods.web]

"main.f" 2 ≡

```
program calcDODS    /* TESTBENCH FOR WATER MOLECULE - GENERATING INTEGRALS
        AND CALCULATION OF WFN */
double precision vlist(MAX_CENTRES, 4)
double precision eta(MAX_PRIMITIVES, 5)
integer nfirst(7)
integer nlast(7)
integer ntype(7)
integer ncntr(7)
integer nr(NO_OF_TYPES, 3)
integer nbfns, ngmx, noc, nfile, ncmx

integer i, j
double precision vlist1(4), vlist2(4), vlist3(4)
double precision u(5)
double precision hydrE(3), hydrC(3)
double precision oxygE(15), oxygC(15)
double precision S(1000), H(1000), HF(1000), R(1000), Rold(1000)
double precision C(1000), Cbar(1000), V(1000)

double precision crit, damp, E
double precision epsilon(100)
integer scf
integer nelec, nbasis, interp, irite

data nr/0, 1, 0, 0, 2, 0, 0, 1, 1, 0, 3, 0, 0, 2, 2, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 2, 0, 1, 0, 1, 0, 3,
        0, 1, 0, 2, 2, 0, 1, 1, 0, 0, 0, 1, 0, 0, 2, 0, 1, 1, 0, 0, 3, 0, 1, 0, 1, 2, 2, 1/
        /* Coordinates of water in Bohr */
data noc/3/
data vlist1/0.00000000, 0.00000000, 1.79523969, 8.0000/
data vlist2/0.00000000, 0.00000000, 0.00000000, 1.0000/
data vlist3/1.69257088, 0.00000000, 2.39364599, 1.0000/

do i = 1, 4
  vlist(1, i) = vlist1(i)
  vlist(2, i) = vlist2(i)
  vlist(3, i) = vlist3(i)
end do
call print_vlist(vlist, vlist1)    // basis set C Hydrogen 1s
data hydrE/3.42525091, 0.62391373, 0.16885540/
data hydrC/0.15432897, 0.53532814, 0.44463454/    /* C Oxygen 1S 2S 2PX 2PY 2PZ */
data oxygE/130.7093200, 23.8088610, 6.4436083, 5.0331513, 1.1695961, 0.3803890, 5.0331513,
        1.1695961, 0.3803890, 5.0331513, 1.1695961, 0.3803890, 5.0331513, 1.1695961, 0.3803890/
data oxygC/0.15432897, 0.53532814, 0.44463454, −0.09996723, 0.39951283, 0.70011547,
        1.15591627, 0.60768372, 0.39195739, 0.15591627, 0.60768372, 0.39195739, 0.15591627,
        0.60768372, 0.39195739/    /* Do the primitive GTOs in eta */
do i = 1, 3    /* oxygen 1 */
  do j = 1, 15
    eta(j, i) = vlist1(i)
    eta(j, 4) = oxygE(j)
```

```
      eta(j, 5) = oxygC(j)
    end do    /* hydrogen 2 */
    do j = 16, 18
      eta(j, i) = vlist2(i)
      eta(j, 4) = hydrE(j − 15)
      eta(j, 5) = hydrC(j − 15)
    end do    /* hydrogen 3 */
    do j = 19, 21
      eta(j, i) = vlist3(i)
      eta(j, 4) = hydrE(j − 18)
      eta(j, 5) = hydrC(j − 18)
    end do
  end do
  call print_eta(eta, u)    /* specification of contraction */
  data nfirst/1, 4, 7, 10, 13, 16, 19/
  data nlast/3, 6, 9, 12, 15, 18, 21/    /* types of basis functions: s,s,px,py,pz,s,s */
  data ntype/1, 1, 2, 3, 4, 1, 1/    /* nuclear center and basis function */
  data ncntr/1, 1, 1, 1, 1, 2, 3/
  data ngmx/21/
  data nbfns/7/
  data ncmx/3/

  nelec = 10
  nbasis = 7
  irite = 12
  nfile = ERI_UNIT
  crit = 1.00 · 10^−06D
  damp = 0.13 · 10^+00D
  interp = 5

  call genint(ngmx, nbasis, eta, ntype, ncntr, nfirst, nlast, vlist, ncmx, noc, S, H, nfile)

  call shalf(S, R, Cbar, nbasis)

    /* Perform SCF ! */

  STOP
  END

  subroutine print_eta(eta, u)
    double precision eta(MAX_PRIMITIVES, 5), u(5)

    write(*, *) "␣PRIMITIVE␣GTOs"
    do i = 1, 21
      do j = 1, 5
        u(j) = eta(i, j)
      end do
      write(*, 200) (u(j), j = 1, 5)
    end do
200: FORMAT("", (10f10.5))
    return
  end

  subroutine print_vlist(vlist, vlist1)
    double precision vlist(MAX_CENTRES, 4), vlist1(4)

    write(*, *) "␣ATOMIC␣COORDINATES␣AND␣NUCLEAR␣CHARGES"
```

```
      do $i = 1,\ 3$
        do $j = 1,\ 4$
          $vlist1(j) = vlist(i,\ j)$
        end do
        write $(*,\ 200)\ (vlist1(j),\ j = 1,\ 4)$
      end do
200: $FORMAT$("", $(10f10.5)$)
      return
    end
```

[dods.web]

3 SCF

[scf.web]

This is Version 1 of the Hartree-Fock theory implemented for closed shells (RHF) and open shells (UHF-DODS) calculations.

NAME SCF

Perform LCAO-MO-SCF calculation on a molecule.

SYNOPSIS

```
double precision function scf(H, C, nbasis, nelec, nfile,
irite, damp, interp, E, HF, V, R, Rold, Ubar, eps, crit)
integer nbasis, nelec, nfile, irite
double precision damp, E
double precision H(ARB), C(ARB), HF(ARB), V(ARB), R(ARB)
double precision Rold(ARB), Ubar(ARB), eps(ARB)
```

DESCRIPTION

Perform LCAO-MO calculation of either closed-shell RHF type or more general open-shell (real) UHF-DODS type. The method is traditional Roothan repeated diagonalizations of Hartree-Fock matrix until self-consistency is reached:

$$\mathbf{F} \cdot \mathbf{C} = \mathbf{S} \cdot \mathbf{C} \cdot \epsilon$$

ARGUMENTS

H Input: One-electron Hamiltonian of size (`nbasis x nbasis`), i.e., matrix elements of one-electron operator

C Input/Output: An initial MO matrix - it must at least orthigonalize the basis. Normally, it is simply the orthogonalization matrix $\mathbf{S}^{-\frac{1}{2}}$. On output the SCF **C** matrix is placed here.

nbasis Input: the number of *spatial* orbitals in the basis (i.e., half of the number of the spin-basis set functions if `nelec` ≥ 0)

nelec Input: The number of electrons in the system.

nfile The electron-repulsion file unit.

itite Channel number for convergence information or zero if this information is not necessary.

damp Hartree-Fock damping parameter.

interp Interpolation parameter. If 0 no interpolation will be undertaken.

HF Output: for use as the Fock matrix

V Workspace:

R Output: Density matrix

Rold Workspace:

Ubar Workspace:

eps Output: orbital energies (first `nelec` are the occupied orbitals)

E Output: Total HF electronic energy

crit Convergence of the SCF procedure

RETURNS

`YES` if the calculation is converged in `MAX_SCF_ITERATIONS`

`NO` if no convergence is met. Typical usage: `if ( SCF(......) .EQ. YES ) then`

`output succesful calculation`

"main.f" 3 ≡

```
  @m MAX_ITERATIONS 50

    integer function SCF(H, C, nbasis, nelec, nfile, irite, damp, interp, E, HF, V, R, Rold, Cbar,
          epsilon, crit)
      ⟨ Global SCF Declarations #6 ⟩
      ⟨ Internal SCF Declarations #7 ⟩
      ⟨ Select SCF Type #8 ⟩
      ⟨ Set initial matrices and counters #9 ⟩
      do while((icon ≠ 0) ∧ (kount < MAX_ITERATIONS))
        ⟨ Sigle SCF iteration #10 ⟩
      end do
      ⟨ Write the output result #11 ⟩
      ⟨ Formats #12 ⟩
      return
    end
```

[scf.web]

⟨ Global SCF Declarations 3.1 ⟩ ≡

```
    implicit double precision (a − h, o − z)
    integer nbasis, nelec, nfile, irite
    integer interp
    double precision H(ARB), C(ARB), HF(ARB), V(ARB), R(ARB)
    double precision Rold(ARB), Cbar(ARB)
    double precision epsilon(ARB)
    double precision E, damp, crit
```

This code is used in section 3.

[scf.web]

⟨ Internal SCF Declarations 3.2 ⟩ ≡

```
    integer scftype, kount, maxit, nocc, m, mm, i
    double precision term, turm, Rsum
    double precision zero, half
    data zero, half /0.0 · 10^+00D, 0.5 · 10^+00D/
```

This code is used in section 3.

[`scf.web`]

⟨Select SCF Type 3.3⟩ ≡

```
    if (nelec > zero) then    /* closed shell case */
      scftype = CLOSED_SHELL_CALCULATION
      nocc = abs(nelec / 2)
      m = nbasis
    else    /* open shell case */
      scftype = UHF_CALCULATION
      nocc = nelec
      m = nbasis * 2
      call spinor(H, nbasis)
      call spinor(C, nbasis)
    end if
```

This code is used in section 3.

[`scf.web`]

⟨Set initial matrices and counters 3.4⟩ ≡

```
      /* basis set size */
    m = m * m
    do i = 1, mm
      R(i) = zero;
      Rold(i) = zero
    end do
    SCF = YES
    kount = 0
    icon = 100
```

This code is used in section 3.

[scf.web]

⟨ Sigle SCF iteration 3.5 ⟩ ≡

```
$kount = kount + 1$
$E = zero;$
$icon = 0$
do $i = 1,\ mm$
   $HF(i) = H(i)$
   $E = E + R(i) * HF(i)$
enddo
call $scfGR(R,\ HF,\ m,\ nfile)$
do $i = 1,\ mm$
   $E = E + R(i) * HF(i)$
enddo

if ($scftype \equiv UHF\_CALCULATION$)
   $E = half * E$

write ($ERROR\_OUTPUT\_UNIT,\ 200$) $E$

call $gtprd(C,\ HF,\ R,\ m,\ m,\ m)$
call $gmprd(R,\ C,\ HF,\ m,\ m,\ m)$
call $eigen(HF,\ Cbar,\ m)$
do $i = 1,\ m$
   $epsilon(i) = HF(m * (i - 1) + i)$
enddo
call $gmprd(C,\ Cbar,\ V,\ m,\ m,\ m)$
call $scfR(V,\ R,\ m,\ nocc)$
$Rsum = zero$
do $i = 1,\ mm$
   $turm = R(i) - Rold(i)$
   $term = \textbf{dabs}(turm)$
   $Rold(i) = R(i)$
   $C(i) = V(i)$
   if ($term > crit$)
      $icon = icon + 1$
   $Rsum = Rsum + term$
   if ($kount < interp$)
      $R(i) = R(i) - damp * turm$
enddo
```

This code is used in section 3.

[scf.web]

```
⟨ Write the output result 3.6 ⟩ ≡
    write (ERROR_OUTPUT_UNIT, 201) Rsum, icon

    if ((kount ≡ MAX_ITERATIONS) ∧ (icon ≠ 0)) then
      write (ERROR_OUTPUT_UNIT, 204)
    else
      write (ERROR_OUTPUT_UNIT, 202) kount
      write (ERROR_OUTPUT_UNIT, 203) (epsilon(i), i = 1, nocc)
    endif
```

This code is used in section 3.

[scf.web]

```
⟨ Formats 3.7 ⟩ ≡
200: format ("␣Current␣Electronic␣Energy␣=␣", f12.6)
201: format ("␣Convergence␣in␣R␣=␣", f12.5, i6, "␣␣Changing")
202: format ("␣SCF␣converged␣in", i4, "␣iterations")
203: format ("␣Orbital␣Energies␣", (7f10.5))
204: format ("␣SCF␣did␣not␣converged...␣quitting")
```

This code is used in section 3.

[scf.web]

scfGR [scf.web]

"main.f" 3.9 ≡

```
@m locGR(i,j)  (m * (j − 1) + i)

    subroutine scfGR(R, G, m, nfile) double precision R(*), G(*)
    integer m, nfile

    integer mby2
    double precision val
    integer i, j, k, l, is, js, ks, ls, ijs, kls, mu
    integer getint
    double precision zero, one, a, b
    integer pointer, spin, skip
    data  one, zero/1.0 · 10^{+00}D,  0.0 · 10^{+00}D/

    mby2 = m / 2     // rewind nfile
    pointer = 0

    do while(getint(nfile, is, js, ks, ls, mu, val, pointer) ≠ END_OF_FILE)

      ijs = is * (is − 1) / 2 + js
      kls = ks * (ks − 1) / 2 + ls

      do spin = 1, 4
        skip = NO

        select case(spin)
        case(1)
        i = is
        j = js
        k = ks
        l = ls
        case(2)
        i = is + mby2
        j = js + mby2
        k = ks + mby2
        l = ls + mby2
        case(3)
        i = is + mby2
        j = js + mby2
        k = ks
        l = ls
        case(4)
        if (ijs ≡ kls)
          skip = YES
        i = is
        j = js
        k = ks + mby2
        l = ls + mby2
        call order(i, j, k, l) end select

        if (skip ≡ YES)
          cycle

        a = one;
        b = one
        if (spin ≥ 3)
```

```
            $b = zero$
          **call** $GofR(R,\ G,\ m,\ a,\ b,\ i,\ j,\ k,\ l,\ val)$
        **end do**
      **enddo**    /* symmetrize G matrix */
      **do** $i = 1,\ m$
        **do** $j = 1,\ i - 1$
          $ij = locGR(i,\ j)$;
          $ji = locGR(j,\ i)$
          $G(ji) = G(ij)$
        **end do**
      **end do**
      **return end**
```

[scf.web]

GofR

[`scf.web`]

`"main.f"` 15 ≡

```
  subroutine GofR(R, G, m, a, b, i, j, k, l, val)
    double precision R(*), G(*)
    double precision val, a, b
    integer i, j, k, l, m
    integer ij, kl, il, ik, jk, jl
    double precision coul1, coul2, coul3, exch

    ij = locGR(i, j);
    kl = locGR(k, l)
    il = locGR(i, l);
    ik = locGR(i, k)
    jk = locGR(j, k);
    jl = locGR(j, l)
    if (j < k)
      jk = locGR(k, j)
    if (j < l)
      jl = locGR(l, j)

    coul1 = a * R(ij) * val;
    coul2 = a * R(kl) * val;
    exch = b * val

    if (k ≠ l) then
      coul2 = coul2 + coul2
      G(ik) = G(ik) − R(jl) * exch
      if ((i ≠ j) ∧ (j ≥ k))
        G(jk) = G(jk) − R(il) * exch
    end if

    G(il) = G(il) − R(jk) * exch;
    G(ij) = G(ij) + coul2

    if ((i ≠ j) ∧ (j ≥ l))
      G(jl) = G(jl) − R(ik) * exch

    if (ij ≠ kl) then
      coul3 = coul1
      if (i ≠ j)
        coul3 = coul3 + coul1
      if (j ≤ k) then
        G(jk) = G(jk) − R(il) * exch
        if ((i ≠ j) ∧ (i ≤ k))
          G(ik) = G(ik) − R(jl) * exch
        if ((k ≠ l) ∧ (j ≤ l))
          G(jl) = G(jl) − R(ik) * exch
      end if
      G(kl) = G(kl) + coul3
    end if

    return
  end
```

[scf.web]

order

[scf.web]

"main.f" 17 ≡

```
    **subroutine** *order*($i,\ j,\ k,\ l$)
      **integer** $i, j, k, l$
      **integer** *integ*

      $i = \textbf{\textit{abs}}(i);$
      $j = \textbf{\textit{abs}}(j);$
      $k = \textbf{\textit{abs}}(k);$
      $l = \textbf{\textit{abs}}(l)$

      **if** $(i < j)$ **then**
        *integ* $= i$
        $i = j$
        $j =$ *integ*
      **end if**

      **if** $(k < l)$ **then**
        *integ* $= k$
        $k = l$
        $l =$ *integ*
      **end if**

      **if** $((i < k) \mid ((i \equiv k) \wedge (j < l)))$ **then**
        *integ* $= i$
        $i = k$
        $k =$ *integ*
        *integ* $= j$
        $j = l$
        $l =$ *integ*
      **end if**

      **return**
    **end**
```

[scf.web]

scfR [`scf.web`]

"`main.f`" 3.13 ≡

```
    subroutine $scfR(C, R, m, nocc)$
       double precision $C(ARB), R(ARB)$
       integer $m, nocc$

       double precision $suma, zero$
       integer $i, j, k, ij, ji, kk, ik, jk$
       data $zero/0.0 \cdot 10^{+00}$D/

       do $i = 1, m$
          do $j = 1, i$
             $suma = zero$
             do $k = 1, nocc$
                $kk = m * (k - 1)$
                $ik = kk + i$
                $jk = kk + j$
                $suma = suma + C(ik) * C(jk)$
             enddo
             $ij = m * (j - 1) + i$
             $ji = m * (i - 1) + j$
             $R(ij) = suma$
             $R(ji) = suma$
          enddo
       enddo

       return
    end
```

[`scf.web`]

4 INTEGRALS

[`integral.web`]

[`integral.web`]

genoei [`integral.web`] Function to compute the one-electron integrals (overlap, kinetic energy and nuclear attraction). The STRUCTURES and GENOEI manual pages must be consulted for a detailed description of the calling sequence.

The overlap and kinetic energy integrals are expressed in terms of a basic one-dimensional Cartesian overlap component computed by **function** *ovrlap* while the more involved nuclear-attraction integrals are computed as a sum of geometrical factors computed by **subroutine** *aform* and the standard F_ν computed by **function** *fmch*.

`"main.f"` 4.2 ≡

```
  double precision function genoei(i, j, eta, ngmx, nfirst, nlast, ntype, nr, ntmx, vlist, noc,
        ncmx, ovltot, kintot)
    implicit double precision (a − h, o − z)
    integer i, j, ngmx, ncmx, noc, ntmx
    integer nfirst(∗), nlast(∗), ntype(∗), nr(ntmx, 3)
    double precision ovltot, kintot
    double precision eta(MAX_PRIMITIVES, 5), vlist(MAX_CENTRES, 4)

      /∗ Insert delarations which are purely local to genoei ∗/

    ⟨ genoei local declarations #25 ⟩

      /∗ Insert the Factorials ∗/

    ⟨ Factorials #36 ⟩

      /∗ Obtain the powers of x,y,z and summation limits ∗/

    ⟨ One-electron Integer Setup #26 ⟩

      /∗ Inter-nuclear distance ∗/

    $rAB = (eta(iss, 1) - eta(jss, 1))^2 + (eta(iss, 2) - eta(jss, 2))^2 + (eta(iss, 3) - eta(jss, 3))^2$

      /∗ Initialise all accumulators ∗/

    genoei = zero
    totnai = zero
    kintot = zero
    ovltot = zero

      /∗ Now start the summations over the contracted GTFs ∗/

    do irun = iss, il     /∗ start of "i" contraction ∗/

      do jrun = jss, jl     /∗ start of "j" contraction ∗/

        ⟨ Compute PA #38 ⟩     /∗ Use the Gaussian-product theorem to find $\vec{P}$ ∗/

        ⟨ Overlap Components #27 ⟩

        ovltot = ovltot + anorm ∗ bnorm ∗ ovl     /∗ accumulate Overlap ∗/

        ⟨ Kinetic Energy Components #29 ⟩

        kintot = kintot + anorm ∗ bnorm ∗ kin     /∗ accumulate Kinetic energy ∗/

          /∗ now the nuclear attraction integral ∗/
        tnai = zero

        ⟨ Form fj #30 ⟩     /∗ Generate the required $f_j$ coefficients ∗/
        do n = 1, noc    /* loop over nuclei */
          pn = zero    /* Initialise current contribution */
            /* Get the attracting-nucleus information; co-ordinates */
          ⟨ Nuclear data #33 ⟩
          t = t1 * pcsq
          call auxg(m, t, g)    /* Generate all the $F_\nu$ required */
          ⟨ Form As #31 ⟩    /* Generate the geometrical A-factors */
            /* Now sum the products of the geometrical A-factors and the $F_\nu$ */
          do ii = 1, imax
            do jj = 1, jmax
              do kk = 1, kmax
                nu = ii + jj + kk − 2
                pn = pn + Airu(ii) * Ajsv(jj) * Aktw(kk) * g(nu)
              end do
            end do
          end do
          tnai = tnai − pn * vlist(n, 4)    /* Adtotal multiplied by currentrrent charge */
        end do    /* end of loop over nuclei */
        totnai = totnai + prefa * tnai
      end do    /* end of "j" contraction */
    end do    /* end of "i" contraction */
    genoei = totnai + kintot    /* "T + V" */
    return
  end
```

[integral.web] These are the declarations which are local to *genoei*, working space *etc.*

⟨ genoei local declarations 4.3 ⟩ ≡

```
  double precision Airu(10), Ajsv(10), Aktw(10)
  double precision p(3), sf(10, 3), tf(20)
  double precision fact(20), g(50)
  double precision kin
  data zero, one, two, half, quart/0.0 · 10^00D, 1.0 · 10^00D, 2.0 · 10^00D, 0.5 · 10^00D, 0.25 · 10^00D/
  data pi/3.141592653589 · 10^00D/
```

This code is used in section 4.2.

[`integral.web`] Get the various powers of x, y and z required from the data structures and obtain the contraction limits etc.

⟨ One-electron Integer Setup 4.4 ⟩ ≡

```
    $ityp = ntype(i)$;
    $jtyp = ntype(j)$
    $l1 = nr(ityp, 1)$;
    $m1 = nr(ityp, 2)$;
    $n1 = nr(ityp, 3)$
    $l2 = nr(jtyp, 1)$;
    $m2 = nr(jtyp, 2)$;
    $n2 = nr(jtyp, 3)$
    $imax = l1 + l2 + 1$;
    $jmax = m1 + m2 + 1$;
    $kmax = n1 + n2 + 1$
    $maxall = imax$
    **if** $(maxall < jmax)$
       $maxall = jmax$
    **if** $(maxall < kmax)$
       $maxall = kmax$
    **if** $(maxall < 2)$
       $maxall = 2$     /* when all functions are "s" type */
    $iss = nfirst(i)$;
    $il = nlast(i)$
    $jss = nfirst(j)$;
    $jl = nlast(j)$
```

This code is used in section 4.2.

[`integral.web`] This simple code gets the Cartesian overlap components and assembles the total integral. It also computes the overlaps required to calculate the kinetic energy integral used in a later module.

```
⟨ Overlap Components 4.5 ⟩ ≡
    $prefa = two * prefa$
    $expab = \textbf{dexp}(-aexp * bexp * rAB / t1)$
    $s00 = (pi / t1)^{1.5} * expab$
    $dum = one;$
    $tf(1) = one;$
    $del = half / t1$
    do $n = 2, maxall$
      $tf(n) = tf(n-1) * dum * del$
      $dum = dum + two$
    end do

    $ox0 = ovrlap(l1, l2, pax, pbx, tf)$
    $oy0 = ovrlap(m1, m2, pay, pby, tf)$
    $oz0 = ovrlap(n1, n2, paz, pbz, tf)$
    $ox2 = ovrlap(l1, l2 + 2, pax, pbx, tf)$
    $oxm2 = ovrlap(l1, l2 - 2, pax, pbx, tf)$
    $oy2 = ovrlap(m1, m2 + 2, pay, pby, tf)$
    $oym2 = ovrlap(m1, m2 - 2, pay, pby, tf)$
    $oz2 = ovrlap(n1, n2 + 2, paz, pbz, tf)$
    $ozm2 = ovrlap(n1, n2 - 2, paz, pbz, tf)$
    $ov0 = ox0 * oy0 * oz0;$
    $ovl = ov0 * s00$
    $ov1 = ox2 * oy0 * oz0;$
    $ov4 = oxm2 * oy0 * oz0$
    $ov2 = ox0 * oy2 * oz0;$
    $ov5 = ox0 * oym2 * oz0$
    $ov3 = ox0 * oy0 * oz2;$
    $ov6 = ox0 * oy0 * ozm2$
```

This code is used in section 4.2.

ovrlap

[`integral.web`] One-dimensional Cartesian overlap. This function uses the precomputed factors in *tf* to evaluate the simple Cartesian components of the overlap integral which must be multiplied together to form the total overlap integral.

`"main.f"` 27 ≡

```
double precision function $ovrlap(l1, l2, pax, pbx, tf)$
  implicit double precision $(a - h, o - z)$
  integer $l1, l2$
  double precision $pax, pbx$
  double precision $tf(*)$
      /* pre-computed exponent and double factorial factors: tf(i+1) = (2i-1)!/(2**i*(A+B)**i) */
  double precision $zero, one, dum$
  data $zero, one / 0.0 \cdot 10^{00}\mathtt{D},\ 1.0 \cdot 10^{00}\mathtt{D}/$
  if $((l1 < 0) \mid (l2 < 0))$ then
    $ovrlap = zero$
    return
  end if
  if $((l1 \equiv 0) \wedge (l2 \equiv 0))$ then
    $ovrlap = one$
    return
  end if
  $dum = zero;$
  $maxkk = (l1 + l2) / 2 + 1$
  do $kk = 1, maxkk$
    $dum = dum + tf(kk) * fj(l1, l2, 2 * kk - 2, pax, pbx)$
  end do
  $ovrlap = dum$
  return
end
```

[`integral.web`] Use the previously-computed overlap components to generate the Kinetic energy components and hence the total integral.

⟨ Kinetic Energy Components 4.6 ⟩ ≡

```
$xl = \textbf{dfloat}(l2 * (l2 - 1));$
$xm = \textbf{dfloat}(m2 * (m2 - 1))$
$xn = \textbf{dfloat}(n2 * (n2 - 1));$
$xj = \textbf{dfloat}(2 * (l2 + m2 + n2) + 3)$
$kin = s00 * (bexp * (xj * ov0 - two * bexp * (ov1 + ov2 + ov3)) - half * (xl * ov4 + xm * ov5 + xn * ov6))$
```

This code is used in section 4.2.

[`integral.web`] Form the f_j coefficients needed for the nuclear attraction integral.

```
⟨ Form fj 4.7 ⟩ ≡
    m = imax + jmax + kmax − 2
    do n = 1, imax
        sf(n, 1) = fj(l1, l2, n − 1, pax, pbx)
    end do

    do n = 1, jmax
        sf(n, 2) = fj(m1, m2, n − 1, pay, pby)
    end do

    do n = 1, kmax
        sf(n, 3) = fj(n1, n2, n − 1, paz, pbz)
    end do
```

This code is used in section 4.2.

[`integral.web`] Use *aform* to compute the required *A*-factors for each Cartesian component.

```
⟨ Form As 4.8 ⟩ ≡
    epsi = quart / t1
    do ii = 1, 10
        Airu(ii) = zero
        Ajsv(ii) = zero
        Aktw(ii) = zero
    end do

    call aform(imax, sf, fact, cpx, epsi, Airu, 1)    /* form $A_{i,r,u}$ */
    call aform(jmax, sf, fact, cpy, epsi, Ajsv, 2)    /* form $A_{j,s,v}$ */
    call aform(kmax, sf, fact, cpz, epsi, Aktw, 3)    /* form $A_{k,t,w}$ */
```

This code is used in section 4.2.

aform

[`integral.web`] Compute the nuclear-attraction A factors. These quantitities arise from the components of the three position vectors of the two basis functions and the attracting centre with respect to the centre of the product Gaussian. There is one of these for each of the three dimensions of Cartesian space; a typical one (the x component) is:

$$A_{\ell,r,i}(\ell_1,\ell_2,\vec{A}_x,\vec{B}_x,\vec{C}_x,\gamma) = (-1)^\ell f_\ell(\ell_1,\ell_2,\vec{PA}_x,\vec{PB}_x)\frac{(-1)^i\ell!\vec{PC}_x^{\ell-2r-2i}\epsilon^{r+i}}{r!i!(\ell-2r-2i)!}$$

`"main.f"` 31 ≡

```
subroutine aform(imax, sf, fact, cpx, epsi, Airu, xyorz)
  implicit double precision (a − h, o − z)
  integer imax, xyorz
  double precision Airu(∗), fact(∗), sf(10, ∗)

  double precision one
  data one/1.0 · 10^00D/

  do i = 1, imax
    ai = (−one)^(i−1) ∗ sf(i, xyorz) ∗ fact(i)
    irmax = (i − 1) / 2 + 1
    do ir = 1, irmax
      irumax = irmax − ir + 1
      do iru = 1, irumax
        iq = ir + iru − 2
        ip = i − 2 ∗ iq − 1
        at5 = one
        if (ip > 0)
          at5 = cpx^ip
        tiru = ai ∗ (−one)^(iru−1) ∗ at5 ∗ epsi^iq / (fact(ir) ∗ fact(iru) ∗ fact(ip + 1))
        nux = ip + iru
        Airu(nux) = Airu(nux) + tiru
      end do
    end do
  end do

  return
end
```

[`integral.web`] Get the co-ordinates of the attracting nucleus with respect to $\vec{P}$.

⟨ Nuclear data 4.9 ⟩ ≡

```
cpx = p(1) − vlist(n, 1)
cpy = p(2) − vlist(n, 2)
cpz = p(3) − vlist(n, 3)
pcsq = cpx ∗ cpx + cpy ∗ cpy + cpz ∗ cpz
```

This code is used in section 4.2.

generi [`integral.web`] The general electron-repulsion integral formula for contracted Gaussian basis functions. The STRUCTURES and GENERI manual pages must be consulted for a detailed description of the calling sequence.

`"main.f"` 4.10 ≡

```
double precision function generi(i, j, k, l, xyorz, eta, ngmx, nfirst, nlast, ntype, nr, ntmx)
  implicit double precision (a − h, o − z)
  integer i, j, k, l, xyorz, ngmx, ntmx
  double precision eta(MAX_PRIMITIVES, 5)
  integer nfirst(∗), nlast(∗), ntype(∗), nr(ntmx, 3)

    /∗ Variables local to the function ∗/

  ⟨ generi local declarations #35 ⟩

    /∗ Insert the data statement for the factorials ∗/

  ⟨ Factorials #36 ⟩

    /∗ Get the various integers from the data structures for the summation limits, Cartesian monomial
      powers etc. from the main integer data structures ∗/

  ⟨ Two-electron Integer Setup #37 ⟩

    /∗ Two internuclear distances this time ∗/

  $rAB = (eta(is, 1) − eta(js, 1))^2 + (eta(is, 2) − eta(js, 2))^2 + (eta(is, 3) − eta(js, 3))^2$
  $rCD = (eta(ks, 1) − eta(ls, 1))^2 + (eta(ks, 2) − eta(ls, 2))^2 + (eta(ks, 3) − eta(ls, 3))^2$

    /∗ Initialise the accumulator ∗/

  generi = zero

    /∗ Now the real work, begin the four contraction loops ∗/

  do irun = is, il     /∗ start of "i" contraction ∗/

    do jrun = js, jl     /∗ start of "j" contraction ∗/

        /∗ Get the data for the two basis functions referring to electron 1; orbital exponents and
          Cartesian co-ordinates and hence compute the vector $\vec{P}$ and the components of $\vec{PA}$ and
          $\vec{PB}$ ∗/

      ⟨ Compute PA #38 ⟩

        /∗ Use function fj and subroutine theta to calculate the geometric factors arising from the
          expansion of the product of Cartesian monomials for the basis functions of electron 1 ∗/

      ⟨ Thetas for electron 1 #40 ⟩

      do krun = ks, kl     /∗ start of "k" contraction ∗/
        do lrun = ls, ll     /∗ start of "l" contraction ∗/
          eribit = zero     /∗ local accumulator ∗/

            /∗ Get the data for the two basis functions referring to electron 2; orbital exponents and
              Cartesian co-ordinates and hence compute the vector $\vec{Q}$ and the components of $\vec{QC}$
              and $\vec{QD}$ ∗/

          ⟨ Compute QC #39 ⟩

          w = pi / (t1 + t2)
```

```
            /* Repeat the use of **function** $fj$ to obtain the geometric factors arising from the
               expansion of Cartesian monomials for the basis functions of electron 2 */

          ⟨ fj for electron 2 #41 ⟩

          call $auxg(m,\ t,\ g)$     /* Obtain the $F_\nu$ by recursion */

            /* Now use the pre-computed $\theta$ factors for both electron distributions to form the overall
               $B$ factors */

          ⟨ Form Bs #42 ⟩

            /* Form the limits and add up all the bits, the products of $x$, $y$ and $z$ related B factors
               and the $F_\nu$ */

          $jt1 = i1max + i2max - 1$
          $jt2 = j1max + j2max - 1$
          $jt3 = k1max + k2max - 1$

          do $ii = 1,\ jt1$
            do $jj = 1,\ jt2$
              do $kk = 1,\ jt3$
                $nu = ii + jj + kk - 2$
                if $(xyorz \neq 0)$
                  $nu = nu + 1$

                  /* $eribit$ is a repulsion integral over primitive GTFs */

                $eribit = eribit + g(nu) * bbx(ii) * bby(jj) * bbz(kk)$

              end do
            end do
          end do

            /* Now accumulate the primitive integrals into the integral over contracted GTFs
               including some constant factors and contraction coefficients */

          $generi = generi + prefa * prefc * eribit * \textbf{dsqrt}(w)$

        end do    /* end of "l" contraction loop */
      end do    /* end of "k" contraction loop */
    end do    /* end of "j" contraction loop */
  end do    /* end of "i" contraction loop */

  if $(xyorz \equiv 0)$
    $generi = generi * two$
  return
end
```

[integral.web] Here are the local declarations (workspoace *etc.*) for the two-electron main function *generi*.

⟨ generi local declarations 4.11 ⟩ ≡

```
    double precision p(3), q(3), ppx(20), ppy(20), ppz(20)
    double precision bbx(20), bby(20), bbz(20), sf(10, 6)
    double precision xleft(5, 10), yleft(5, 10), zleft(5, 10)
    double precision r(3), fact(20), g(50)
    data zero, one, two, half/0.0·10^00D, 1.0·10^00D, 2.0·10^00D, 0.5·10^00D/
    data pi/3.141592653589·10^00D/
```

This code is used in section 4.10.

[integral.web] These numbers are the first 20 factorials *fact*(*i*) contains $(i-1)!$.

⟨ Factorials 4.12 ⟩ ≡

```
    data fact/1.0·10^00D, 1.0·10^00D, 2.0·10^00D, 6.0·10^00D, 24.0·10^00D, 120.0·10^00D, 720.0·10^00D,
          5040.0·10^00D, 40320.0·10^00D, 362880.0·10^00D, 3628800.0·10^00D, 39916800.0·10^00D,
          479001600.0·10^00D, 6227020800.0·10^00D, 6 ∗ 0.0·10^00D/
```

This code is used in sections 4.2, 4.10, and #44.

[integral.web] This tedious code extracts the (integer) setup data; the powers of x, y and z in each of the Cartesian monomials of each of the four basis functions and the limits of the contraction in each case.

⟨ Two-electron Integer Setup 4.13 ⟩ ≡

```
    ityp = ntype(i)
    jtyp = ntype(j)
    ktyp = ntype(k)
    ltyp = ntype(l)
    l1 = nr(ityp, 1)
    m1 = nr(ityp, 2)
    n1 = nr(ityp, 3)
    l2 = nr(jtyp, 1)
    m2 = nr(jtyp, 2)
    n2 = nr(jtyp, 3)
    l3 = nr(ktyp, 1)
    m3 = nr(ktyp, 2)
    n3 = nr(ktyp, 3)
    l4 = nr(ltyp, 1)
    m4 = nr(ltyp, 2)
    n4 = nr(ltyp, 3)
    is = nfirst(i)
    il = nlast(i)
    js = nfirst(j)
    jl = nlast(j)
    ks = nfirst(k)
    kl = nlast(k)
    ls = nfirst(l)
    ll = nlast(l)
```

This code is used in section 4.10.

[integral.web] Use the Gaussian Product Theorem to find the position vector $\vec{P}$, of the product of the two Gaussian exponential factors of the basis functions for electron 1.

⟨ Compute PA 4.14 ⟩ ≡
```
    $aexp = eta(irun, 4);$
    $anorm = eta(irun, 5)$
    $bexp = eta(jrun, 4);$
    $bnorm = eta(jrun, 5)$

      /* aexp and bexp are the primitive GTF exponents for GTF irun and jrun, anorm and bnorm are
         the corresponding contraction coefficients bundled up into prefa */

    $t1 = aexp + bexp;$
    $deleft = one / t1$

    $p(1) = (aexp * eta(irun, 1) + bexp * eta(jrun, 1)) * deleft$
    $p(2) = (aexp * eta(irun, 2) + bexp * eta(jrun, 2)) * deleft$
    $p(3) = (aexp * eta(irun, 3) + bexp * eta(jrun, 3)) * deleft$

    $pax = p(1) - eta(irun, 1)$
    $pay = p(2) - eta(irun, 2)$
    $paz = p(3) - eta(irun, 3)$

    $pbx = p(1) - eta(jrun, 1)$
    $pby = p(2) - eta(jrun, 2)$
    $pbz = p(3) - eta(jrun, 3)$

    $prefa = \mathbf{dexp}(-aexp * bexp * rAB / t1) * pi * anorm * bnorm / t1$
```

This code is used in sections 4.2 and 4.10.

[`integral.web`] Use the Gaussian Product Theorem to find the position vector $\vec{Q}$, of the product of the two Gaussian exponential factors of the basis functions for electron 2.

⟨ Compute QC 4.15 ⟩ ≡

```
    cexpp = eta(krun, 4);
    cnorm = eta(krun, 5)
    dexpp = eta(lrun, 4);
    dnorm = eta(lrun, 5)

      /* cexp and dexp are the primitive GTF exponents for GTF krun and lrun, cnorm and dnorm are
         the corresponding contraction coefficients bundled up into prefc */

    t2 = cexpp + dexpp
    t2m1 = one / t2
    fordel = t2m1 + deleft

    q(1) = (cexpp * eta(krun, 1) + dexpp * eta(lrun, 1)) * t2m1
    q(2) = (cexpp * eta(krun, 2) + dexpp * eta(lrun, 2)) * t2m1
    q(3) = (cexpp * eta(krun, 3) + dexpp * eta(lrun, 3)) * t2m1

    qcx = q(1) - eta(krun, 1)
    qcy = q(2) - eta(krun, 2)
    qcz = q(3) - eta(krun, 3)

    qdx = q(1) - eta(lrun, 1)
    qdy = q(2) - eta(lrun, 2)
    qdz = q(3) - eta(lrun, 3)

    r(1) = p(1) - q(1)
    r(2) = p(2) - q(2)
    r(3) = p(3) - q(3)

    t = (r(1) * r(1) + r(2) * r(2) + r(3) * r(3)) / fordel
    prefc = exp(-cexpp * dexpp * rCD / t2) * pi * cnorm * dnorm / t2
```

This code is used in section 4.10.

[`integral.web`] The series of terms arising from the expansion of the Cartesian monomials like $(x - PA)^{\ell_1}(x - PB)^{\ell_2}$ are computed by first forming the f_j and hence the θs.

⟨ Thetas for electron 1 4.16 ⟩ ≡

```
    $i1max = l1 + l2 + 1$
    $j1max = m1 + m2 + 1$
    $k1max = n1 + n2 + 1$

    $mleft = i1max + j1max + k1max$

    do $n = 1,\ i1max$
       $sf(n,\ 1) = fj(l1,\ l2,\ n - 1,\ pax,\ pbx)$
    end do

    do $n = 1,\ j1max$
       $sf(n,\ 2) = fj(m1,\ m2,\ n - 1,\ pay,\ pby)$
    end do

    do $n = 1,\ k1max$
       $sf(n,\ 3) = fj(n1,\ n2,\ n - 1,\ paz,\ pbz)$
    end do

    call $theta(i1max,\ sf,\ 1,\ fact,\ t1,\ xleft)$
    call $theta(j1max,\ sf,\ 2,\ fact,\ t1,\ yleft)$
    call $theta(k1max,\ sf,\ 3,\ fact,\ t1,\ zleft)$
```

This code is used in section 4.10.

[`integral.web`] The series of terms arising from the expansion of the Cartesian monomials like $(x - QC)^{\ell_3}(x - QD)^{\ell_4}$ are computed by forming the f_j and storing them in the array *sf* for later use by *bform*.

⟨ fj for electron 2 4.17 ⟩ ≡

```
    $i2max = l3 + l4 + 1$
    $j2max = m3 + m4 + 1$
    $k2max = n3 + n4 + 1$

    $twodel = half * fordel$
    $delta = half * twodel$

    do $n = 1,\ i2max$
       $sf(n,\ 4) = fj(l3,\ l4,\ n - 1,\ qcx,\ qdx)$
    end do

    do $n = 1,\ j2max$
       $sf(n,\ 5) = fj(m3,\ m4,\ n - 1,\ qcy,\ qdy)$
    end do

    do $n = 1,\ k2max$
       $sf(n,\ 6) = fj(n3,\ n4,\ n - 1,\ qcz,\ qdz)$
    end do

    $m = mleft + i2max + j2max + k2max + 1$
```

This code is used in section 4.10.

[`integral.web`] In the central inner loops of the four contractions, use the previously- computed θ factors to form the combined geometrical B factors.

⟨ Form Bs 4.18 ⟩ ≡

```
$ppx(1) = one;$
$bbx(1) = zero$
$ppy(1) = one;$
$bby(1) = zero$
$ppz(1) = one;$
$bbz(1) = zero$

$jt1 = i1max + i2max$
do $n = 2, jt1$
   $ppx(n) = -ppx(n-1) * r(1)$
   $bbx(n) = zero$
end do

$jt1 = j1max + j2max$
do $n = 2, jt1$
   $ppy(n) = -ppy(n-1) * r(2)$
   $bby(n) = zero$
end do

$jt1 = k1max + k2max$
do $n = 2, jt1$
   $ppz(n) = -ppz(n-1) * r(3)$
   $bbz(n) = zero$
end do

call $bform(i1max, i2max, sf, 1, fact, xleft, t2, delta, ppx, bbx, xyorz)$
call $bform(j1max, j2max, sf, 2, fact, yleft, t2, delta, ppy, bby, xyorz)$
call $bform(k1max, k2max, sf, 3, fact, zleft, t2, delta, ppz, bbz, xyorz)$
```

This code is used in section 4.10.

[`integral.web`]

fj [`integral.web`] This is the function to evaluate the coefficient of x^j in the expansion of

$$(x+a)^\ell (x+b)^m$$

The full expression is

$$f_j(\ell, m, a, b) = \sum_{k=max(0,j-m)}^{min(j,\ell} \binom{\ell}{k}\binom{m}{j-k} a^{\ell-k} b^{m+k-j}$$

The function must take steps to do the right thing for 0.0^0 when it occurs.

`"main.f"` 4.20 ≡

```
**double precision function** $fj(l,\ m,\ j,\ a,\ b)$
    **implicit double precision** $(a-h,\ o-z)$
    **integer** $l$, $m$, $j$
    **double precision** $a$, $b$

    **double precision** $sum$, $term$, $aa$, $bb$
    **integer** $i$, $imax$, $imin$
    **double precision** $fact(20)$

    ⟨ Factorials 4.12 ⟩

    $imax = \mathbf{min}(j,\ l)$
    $imin = \mathbf{max}(0,\ j-m)$

    $sum = 0.0 \cdot 10^{00}$D
    **do** $i = imin,\ imax$

        $term = fact(l+1) * fact(m+1) \,/\, (fact(i+1) * fact(j-i+1))$
        $term = term \,/\, (fact(l-i+1) * fact(m-j+i+1))$
        $aa = 1.0 \cdot 10^{00}$D;
        $bb = 1.0 \cdot 10^{00}$D
        **if** $((l-i) \neq 0)$
            $aa = a^{l-i}$

        **if** $((m+i-j) \neq 0)$
            $bb = b^{m+i-j}$

        $term = term * aa * bb$
        $sum = sum + term$

    **end do**

    $fj = sum$

    **return**
**end**
```

[`integral.web`]

theta

[integral.web] Computation of all the θ factors required from one basis-function product; any one of them is given by

$$\theta(j, \ell_1, \ell_2, a, b, r, \gamma) = f_j(\ell_1, \ell_2, a, b)\frac{j!\gamma^{r-j}}{r!(j-2r)!}$$

The f_j are computed in the body of *generi* and passed to this routine in *sf*, the particular ones to use are in *sf*$(*,$ *isf*$)$. They are stored in *xleft*, *yleft* and *zleft* because they are associated with electron 1 (the left-hand factor in the integrand as it is usually written $(ij, k\ell)$).

"main.f" 45 $\equiv$

```
    subroutine theta(i1max, sf, isf, fact, t1, xleft)
      implicit double precision (a − h, o − z)
      integer i1max, isf
      double precision t1
      double precision sf(10, ∗), fact(∗), xleft(5, ∗)
      integer i1, ir1, ir1max, jt2
      double precision zero, sfab, bbb
      data  zero/0.0 · 10^00 D/
      do i1 = 1, 10
        do ir1 = 1, 5
          xleft(ir1, i1) = zero
        end do
      end do
      do 100 i1 = 1, i1max
        sfab = sf(i1, isf)
        if (sfab ≡ zero)
          go to 100
        ir1max = (i1 − 1) / 2 + 1
        bbb = sfab ∗ fact(i1) / t1^(i1 −1)
        do ir1 = 1, ir1max
          jt2 = i1 + 2 − ir1 − ir1
          xleft(ir1, i1) = bbb ∗ (t1^(ir1 −1)) / (fact(ir1) ∗ fact(jt2))
        end do
100: continue
      return
    end
```

[integral.web]

bform

[`integral.web`] Use the pre-computed f_j and θ to form the "B " factors, the final geometrical expansion coefficients arising from the products of Cartesian monomials. Any one of them is given by

$$\begin{aligned} &B_{\ell,\ell',r_1,r_2,i}(\ell_1,\ell_2,\vec{A}_x,\vec{B}_x,\vec{P}_x,\gamma_1;\ell_3,\ell_4,\vec{C}_x,\vec{D}_x,\vec{Q}_x,\gamma_2) \\ &= (-1)^{\ell'}\theta(\ell,\ell_1,\ell_2,\vec{PA}_x,\vec{PB}_x,r,\gamma_1)\theta(\ell',\ell_3,\ell_4,\vec{QC}_x,\vec{QD}_x,r',\gamma_2) \\ &\times\frac{(-1)^i(2\delta)^{2(r+r')}(\ell+\ell'-2r-2r')!\delta^i\bar{p}_x^{\ell+\ell'-2(r+r'+i)}}{(4\delta)^{\ell+\ell'}i![\ell+\ell'-2(r+r'+i)]!}\end{aligned}$$

`"main.f"` 47 $\equiv$

```
subroutine bform(i1max, i2max, sf, isf, fact, xleft, t2, delta, ppx, bbx, xyorz)
  implicit double precision (a − h, o − z)
  integer i1max, i2max, isf
  double precision fact(∗), sf(10, ∗), xleft(5, ∗), bbx(∗), ppx(20)
  double precision delta
  integer xyorz, itab

  double precision zero, one, two, twodel, fordel, sfab, sfcd
  double precision bbc, bbd, bbe, bbf, bbg, ppqq
  integer i1, i2, jt1, jt2, ir1max, ir2max
  data zero, one, two/0.0 · 10^00D, 1.0 · 10^00D, 2.0 · 10^00D/

  itab = 0

  if (xyorz ≡ isf)
    itab = 1

  twodel = two ∗ delta;
  fordel = two ∗ twodel

  do 200 i1 = 1, i1max

    sfab = sf(i1, isf)
    if (sfab ≡ zero)
      go to 200
    ir1max = (i1 − 1) / 2 + 1

    do 210 i2 = 1, i2max

      sfcd = sf(i2, isf + 3)
      if (sfcd ≡ zero)
        go to 210
      jt1 = i1 + i2 − 2
      ir2max = (i2 − 1) / 2 + 1
      bbc = ((−one)^(i2−1)) ∗ sfcd ∗ fact(i2) / (t2^(i2−1) ∗ (fordel^jt1))

      do 220 ir1 = 1, ir1max

        jt2 = i1 + 2 − ir1 − ir1
        bbd = bbc ∗ xleft(ir1, i1)
        if (bbd ≡ zero)
          go to 220

        do 230 ir2 = 1, ir2max
```

```
            $jt3 = i2 + 2 - ir2 - ir2$
            $jt4 = jt2 + jt3 - 2$
            $irumax = (jt4 + itab) / 2 + 1$
            $jt1 = ir1 + ir1 + ir2 + ir2 - 4$
            $bbe = bbd * (t2^{ir2-1}) * (twodel^{jt1}) * fact(jt4 + 1) / (fact(ir2) * fact(jt3))$
            **do** 240 $iru = 1,\ irumax$
              $jt5 = jt4 - iru - iru + 3$
              $ppqq = ppx(jt5)$
              **if** $(ppqq \equiv zero)$
                **go to** 240
              $bbf = bbe * ((-delta)^{iru-1}) * ppqq / (fact(iru) * fact(jt5))$
              $bbg = one$
              **if** $(itab \equiv 1)$ **then**
                $bbg = \textbf{dfloat}(jt4 + 1) * ppx(2) / (delta * \textbf{dfloat}(jt5))$
              **end if**
              $bbf = bbf * bbg$
              $nux = jt4 - iru + 2$
              $bbx(nux) = bbx(nux) + bbf$
        240: **continue**
      230: **continue**
    220: **continue**
  210: **continue**
200: **continue**
    **return**
  **end**
```

[integral.web]

auxg [`integral.web`] Find the maximum value of F_ν required, use *fmch* to compute it and obtain all the lower F_ν by downward recursion.

$$F_{\nu-1}(x) = \frac{\exp(-x) + 2xF_\nu(x)}{2\nu - 1}$$

`"main.f"` 4.24 ≡

```
  subroutine auxg(mmax, x, g)
    implicit double precision (a − h, o − z)
    integer mmax
    double precision x, g(∗)
    double precision fmch
    double precision two, y
    integer mp1mx, mp1, md, mdm
    data two/2.0 · 10^00 D/
    y = dexp(−x)
    mp1mx = mmax + 1
    g(mp1mx) = fmch(mmax, x, y)
    if (mmax < 1)
      go to 303    /∗ just in case! ∗/
      /∗ Now do the recursion downwards ∗/
    do mp1 = 1, mmax
      md = mp1mx − mp1
      mdm = md − 1
      g(md) = (two ∗ x ∗ g(md + 1) + y) / dfloat(2 ∗ mdm + 1)
    end do
303: return
  end
```

[`integral.web`]

fmch

[`integral.web`] This code is for the oldest and most general and reliable of the methods of computing

$$F_\nu(x) = \int_0^1 t^{2\nu} \exp(-xt^2)dt \tag{1}$$

One of two possible series expansions is used depending on the value of x.

For $x \le 10$ (Small x Case) the (potentially) infinite series

$$F_\nu(x) = \frac{1}{2}\exp(-x)\sum_{i=0}^{\infty} \frac{\Gamma(\nu+\frac{1}{2})}{\Gamma(\nu+i+\frac{3}{2})} x^i \tag{2}$$

is used.

The series is truncated when the value of terms falls below 10^{-8}. However, if the series seems to be becoming unreasonably long before this condition is reached (more than 50 terms), the evaluation is stopped and the function aborted with an error message on *ERROR_OUTPUT_UNIT*.

If $x > 10$ (Large x Case) a different series expansion is used:

$$F_\nu(x) = \frac{\Gamma(\nu+\frac{1}{2})}{2x^{\nu+\frac{1}{2}}} - \frac{1}{2}\exp(-x)\sum_{i=0}^{\infty} \frac{\Gamma(\nu+\frac{1}{2})}{\Gamma(\nu-i+\frac{3}{2})} x^{-i} \tag{3}$$

This series, in fact, diverges but it diverges so slowly that the error obtained in truncating it is always less than the last term in the truncated series. Thus, Thus, to obtain a value of the function to the same accuracy as the other series, the expansion is terminated when the last term is less than the same criterion (10^{-8}).

It can be shown that the minimum term is always for i close to $\nu + x$, thus ifthe terms for this value of i are not below the criterion, the series expansion is abandoned, a message output on *ERROR_OUTPUT_UNIT* and the function aborted.

The third argument, y, is $exp(-x)$, since it is assumed that this function will only be used *once* to evaluate the function $F_\nu(x)$ for the maximum value of ν required and other values will be obtained by downward recursion of the form

$$F_{\nu-1}(x) = \frac{\exp(-x) + 2xF_\nu(x)}{2\nu - 1} \tag{4}$$

which also requires the value of $\exp(-x)$ to be available.

NAME

fmch

SYNOPSIS

```
double precision function fmch(nu,x,y)

implicit double precision (a-h,o-z)
double precision x, y
integer nu
```

DESCRIPTION

Computes

$$F_\nu(x) = \int_0^1 t^{2\nu} e^{-xt^2} dt$$

given ν and x. It is used in the evaluation of GTF nuclear attraction and electron-repulsion integrals.

ARGUMENTS

nu Input: The value of ν in the explicit formula above (`integer`)

x Input: x in the formula (`double precision`)

y Input: $\exp(-x)$, assumed to be available.

DIAGNOSTICS

If the relevant series of expansion used do not converge to a tolerance of 10^{-8}, an error message is printed on standard output and the computation aborted.

```
"main.f" 51 ≡
  double precision function fmch(nu, x, y)
    ⟨Declarations #53⟩   /* First, make the variable declarations */
    ⟨Internal Declarations #54⟩
    m = nu
    a = dfloat(m)
    if (x ≤ ten) then
      ⟨Small x Case #55⟩
    else
      ⟨Large x Case #56⟩
    end if
  end
```

[integral.web] Here are the declarations and **data** statements which are ...

```
⟨ Declarations 4.26 ⟩ ≡
      implicit double precision (a − h, o − z)
      double precision x, y
      integer nu
```

This code is used in section 51.

[integral.web]

```
⟨ Internal Declarations 4.27 ⟩ ≡
      double precision ten, half, one, zero, rootpi4, xd, crit
      double precision term, partialsum
      integer m, i, numberofterms, maxone, maxtwo
      data zero, half, one, rootpi4, ten/$0.0 \cdot 10^{00}$D, $0.5 \cdot 10^{00}$D, $1.0 \cdot 10^{00}$D, $0.88622692 \cdot 10^{00}$D, $10.0 \cdot 10^{00}$D/
         /* crit is required accuracy of the series expansion */
      data crit/$1.0 \cdot 10^{-08}$D/    /* maxone */
      data maxone/50/, maxtwo/200/
```

This code is used in section 51.

[integral.web]

```
⟨ Small x Case 4.28 ⟩ ≡
      a = a + half
      term = one / a
      partialsum = term
      do i = 2, maxone
         a = a + one
         term = term ∗ x / a
         partialsum = partialsum + term
         if (term / partialsum < crit)
            go to 111
      end do
111:  continue
      if (i ≡ maxone) then
         write (ERROR_OUTPUT_UNIT, 200)
  200: format ('i␣>␣50␣in␣fmch')
         STOP
      end if
      fmch = half ∗ partialsum ∗ y
      return
```

This code is used in section 51.

[`integral.web`]

```
⟨ Large x Case 4.29 ⟩ ≡
    $b = a + half$
    $a = a - half$
    $xd = one\ /\ x$
    $approx = rootpi4 * \textbf{dsqrt}(xd) * xd^{m}$
    if $(m > 0)$ then
        do $i = 1,\ m$
            $b = b - one$
            $approx = approx * b$
        end do
    end if
    $fimult = half * y * xd$
    $partialsum = zero$

    if $(fimult \equiv zero)$ then
        $fmch = approx$
        return
    end if

    $fiprop = fimult\ /\ approx$
    $term = one$
    $partialsum = term$
    $numberofterms = maxtwo$
    do $i = 2,\ numberofterms$
        $term = term * a * xd$
        $partialsum = partialsum + term$
        if $(\textbf{dabs}(term * fiprop\ /\ partialsum) \le crit)$ then
            $fmch = approx - fimult * partialsum$
            return
        end if
        $a = a - one$
    end do
    write (ERROR_OUTPUT_UNIT, 201)
201: format ('␣numberofterms␣reached␣in␣fmch')
    STOP
```

This code is used in section 51.

[`integral.web`]

5 INTEGRAL STORAGE AND PROCESSING

[`gints.web`]

[`gints.web`]

getint [gints.web] This function withdraws (ij, kl) two-electron integral from the file.

"main.f" 5.2 ≡

```
integer function getint(file, i, j, k, l, mu, val, pointer)

  integer file, i, j, k, l, mu, pointer
  double precision val
  save

  integer max_pointer, id, iend
  double precision zero
  double precision labels(INT_BLOCK_SIZE), value(INT_BLOCK_SIZE)
  data max_pointer/0/, iend/NOT_LAST_BLOCK/, zero/0.0 · 10^00D/

    /* File must be rewound before first use of this function and pointer must be set to 0 */

  if (pointer ≡ max_pointer) then
    if (iend ≡ LAST_BLOCK) then
      val = zero;
      i = 0;
      j = 0;
      k = 0;
      l = 0
      max_pointer = 0;
      iend = NOT_LAST_BLOCK
      getint = END_OF_FILE
      return
    end if
    read (file) max_pointer, iend, labels, value
    pointer = 0
  end if
  pointer = pointer + 1
  call unpack(labels(pointer), i, j, k, l, mu, id)
  val = value(pointer)
  getint = OK

  return
end
```

[gints.web]

putint [gints.web] This function is just happy.

"main.f" 5.4 ≡

```
      subroutine putint(nfile, i, j, k, l, mu, val, pointer, last)
        implicit double precision (a − h, o − z)
        save

        integer nfile, i, j, k, l, mu, pointer, last
        double precision labels(INT_BLOCK_SIZE), value(INT_BLOCK_SIZE)
        double precision val
        data max_pointer/INT_BLOCK_SIZE/, id/0/     /* id is now unused */

        if (last ≡ ERR)
          go to 100
        iend = NOT_LAST_BLOCK
        if (pointer ≡ max_pointer) then
          write (nfile) pointer, iend, labels, value
          pointer = 0
        end if
        pointer = pointer + 1
        call pack(labels(pointer), i, j, k, l, mu, id)
        value(pointer) = val
        if (last ≡ YES) then
          iend = LAST_BLOCK
          last = ERR
          write (nfile) pointer, iend, labels, value
        end if

  100: return
      end
```

[gints.web]

genint [`gints.web`] This subroutine generates one- and two-electron integrals.

`"main.f"` 5.6 ≡

```
subroutine genint(ngmx, nbfns, eta, ntype, ncntr, nfirst, nlast, vlist, ncmx, noc, S, H, nfile)
      integer ngmx, nbfns, noc, ncmx
double precision eta(MAX_PRIMITIVES, 5), vlist(MAX_CENTRES, 4)
double precision S(ARB), H(ARB)
integer ntype(ARB), nfirst(ARB), nlast(ARB), ncntr(ARB), nfile

integer i, j, k, l, ltop, ij, ji, mu, m, n, jtyp, js, jf, ii, jj
double precision generi, genoei
integer pointer, last
double precision ovltot, kintot
double precision val, crit, alpha, t, t1, t2, t3, sum, pitern
double precision SOO
double precision gtoC(ngmx)
double precision dfact(20)
integer nr(NO_OF_TYPES, 3)
data nr/0, 1, 0, 0, 2, 0, 0, 1, 1, 0, 3, 0, 0, 2, 2, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 2, 0, 1, 0, 1, 0, 3,
      0, 1, 0, 2, 2, 0, 1, 1, 0, 0, 0, 1, 0, 0, 2, 0, 1, 1, 0, 0, 3, 0, 1, 0, 1, 2, 2, 1/
data crit, half, onep5, one, zero/1.0 · 10^-08D, 0.5 · 10^+00D, 1.5 · 10^+00D, 1.0 · 10^+00D, 0.0 · 10^+00D/
data dfact/1.0, 3.0, 15.0, 105.0, 945.0, 10395.0, 135135.0, 2027025.0, 12 * 0.0/

mu = 0

⟨Copy GTO contraction coeffs to gtoC #65⟩

⟨Normalize the primitives #66⟩

  /* one electron integrals */

DO i = 1, nbfns DO j = 1, i
ij = (j − 1) * nbfns + i;
ji = (i − 1) * nbfns + j
H(ij) = genoei(i, j, eta, ngmx, nfirst, nlast, ntype, nr, NO_OF_TYPES, vlist, noc, ncmx, ovltot,
    kintot)
H(ji) = H(ij)
S(ij) = ovltot;
S(ji) = ovltot END DO END DO
write(*, *) "␣ONE␣ELECTRON␣INTEGRALS␣COMPUTED"

rewind nfile;
pointer = 0
last = NO
i = 1;
j = 1;
k = 1;
l = 0

DO 10
  WHILE(next_label(i, j, k, l, nbfns) ≡ YES)
IF(l ≡ nbfns)last = YES
val = generi(i, j, k, l, 0, eta, ngmx, nfirst, nlast, ntype, nr, NO_OF_TYPES) IF(dabs(val) < crit)
    go to 10
CALLputint(nfile, i, j, k, l, mu, val, pointer, last)
10: CONTINUE
```

```
    **return end**
```

[`gints.web`]

⟨ Copy GTO contraction coeffs to gtoC 5.7 ⟩ ≡

```
do $i = 1,\ ngmx$
   $gtoC(i) = eta(i,\ 5)$
end do
```

This code is used in section 5.6.

[gints.web]

⟨ Normalize the primitives 5.8 ⟩ ≡

```
      /* First, normalize the primitives */
   $pitern = 5.568327997 \cdot 10^{+00}\mathtt{D}$    /* pi**1.5 */
   do $j = 1,\ nbfns$
     $jtyp = ntype(j)$;
     $js = nfirst(j)$;
     $jf = nlast(j)$
     $l = nr(jtyp,\ 1)$;
     $m = nr(jtyp,\ 2)$;
     $n = nr(jtyp,\ 3)$
     do $i = js,\ jf$
       $alpha = eta(i,\ 4)$;
       $SOO = pitern * (half\ /\ alpha)^{1.5}$
       $t1 = dfact(l)\ /\ alpha^{l}$
       $t2 = dfact(m)\ /\ alpha^{m}$
       $t3 = dfact(n)\ /\ alpha^{n}$
       $eta(i,\ 5) = one\ /\ \mathbf{dsqrt}(SOO * t1 * t2 * t3)$
     end do
   end do
      /* Now normalize the basis functions */
   do $j = 1,\ nbfns$
     $jtyp = ntype(j)$;
     $js = nfirst(j)$;
     $jf = nlast(j)$
     $l = nr(jtyp,\ 1)$;
     $m = nr(jtyp,\ 2)$;
     $n = nr(jtyp,\ 3)$
     $sum = zero$
     do $ii = js,\ jf$
       do $jj = js,\ jf$
         $t = one\ /\ (eta(ii,\ 4) + eta(jj,\ 4))$
         $SOO = pitern * (t^{onep5}) * eta(ii,\ 5) * eta(jj,\ 5)$
         $t = half * t$
         $t1 = dfact(l)\ /\ t^{l}$
         $t2 = dfact(m)\ /\ t^{m}$
         $t3 = dfact(n)\ /\ t^{n}$
         $sum = sum + gtoC(ii) * gtoC(jj) * SOO * t1 * t2 * t3$
       end do
     end do
     $sum = one\ /\ \mathbf{sqrt}(sum)$
     do $ii = js,\ jf$
       $gtoC(ii) = gtoC(ii) * sum$
     end do
   end do
   do $ii = 1,\ ngmx$
     $eta(ii,\ 5) = eta(ii,\ 5) * gtoC(ii)$
   end do
```

This code is used in section 5.6.

[gints.web]

6 UTILITIES

[utilities.web] The utility functions

[utilities.web]

gtprd [matrix.web]

"main.f" 6.2 ≡

```
@m loch(i,j) (n * (j - 1) + i)


  subroutine gtprd(A, B, R, n, m, l)
    double precision A(ARB), B(ARB)
    double precision R(ARB)
    integer n, m, l

    double precision zero
    integer k, ik, j, ir, ij, ib
    data zero/0.0 · 10^{+00}D/    /* stride counters initialization */

    ir = 0;
    ik = -n
    do k = 1, l
      ij = 0
      ik = ik + m
      do j = 1, m
        ir = ir + 1;
        ib = ik
        R(ir) = zero
        do i = 1, n
          ij = ij + 1;
          ib = ib + 1
          R(ir) = R(ir) + A(ij) * B(ib)
        enddo
      enddo
    enddo

    return
  end
```

[matrix.web]

gmprd [matrix.web]

[matrix.web]

"main.f" 6.5 ≡

```
subroutine $gmprd(A,\ B,\ R,\ n,\ m,\ l)$
  double precision $A(ARB)$, $B(ARB)$
  double precision $R(ARB)$
  integer $n$, $m$, $l$

  double precision $zero$
  integer $k$, $ik$, $j$, $ir$, $ji$, $ib$
  data  $zero/0.0 \cdot 10^{+00}$D/     /* stride counters initialization */

  $ir = 0$;
  $ik = -m$
  do $k = 1,\ l$
    $ik = ik + m$
    do $j = 1,\ n$
      $ir = ir + 1$;
      $ji = j - n$;
      $ib = ik$
      $R(ir) = zero$
      do $i = 1,\ m$
        $ji = ji + n$;
        $ib = ib + 1$
        $R(ir) = R(ir) + A(ji) * B(ib)$
      enddo
    enddo
  enddo

  return
end
```

[matrix.web]

eigen [matrix.web]

[matrix.web]

"main.f" 6.8 ≡

```
subroutine eigen(H, U, n)
  implicit double precision (a − h, o − z)
  double precision H(1), U(1)
  integer n
  data zero, eps, one, two, four, big/0.0 · 10^{+00}D, 1.0 · 10^{−20}D, 1.0 · 10^{+00}D, 2.0 · 10^{+00}D,
        4.0 · 10^{+00}D, 1.0 · 10^{+20}D/    /* Initialize U matrix to unity */
  do i = 1, n
    ii = loch(i, i)
    do j = 1, n
      ij = loch(i, j)
      U(ij) = zero
    end do
    U(ii) = one
  end do    /* start sweep through off-diagonal elements */
  hmax = big
  do 90while(hmax > eps)
    hmax = zero
    do i = 2, n
      jtop = i − 1
      do 10 j = 1, jtop
        ii = loch(i, i);
        jj = loch(j, j)
        ij = loch(i, j);
        ji = loch(j, i)
        hii = H(ii);
        hjj = H(jj);
        hij = H(ij)
        hsq = hij ∗ hij
        if (hsq > hmax)
          hmax = hsq
        if (hsq < eps)
          go to 10
        del = hii − hjj;
        sign = one
        if (del < zero) then
          sign = −one
          del = −del
        end if
        denom = del + dsqrt(del ∗ del + four ∗ hsq)
        tan = two ∗ sign ∗ hij / denom
        c = one / dsqrt(one + tan ∗ tan)
        s = c ∗ tan
        do 20 k = 1, n
          kj = loch(k, j);
          ki = loch(k, i)
          jk = loch(j, k);
          ik = loch(i, k)
          temp = c ∗ U(kj) − s ∗ U(ki)
          U(ki) = s ∗ U(kj) + c ∗ U(ki);
```

```
            $U(kj) = temp$
            if $((i \equiv k) \mid (j \equiv k))$
              go to 20     /* update the parts of H matrix affected by a rotation */
            $temp = c * H(kj) - s * H(ki)$
            $H(ki) = s * H(kj) + c * H(ki)$
            $H(kj) = temp$;
            $H(ik) = H(ki)$;
            $H(jk) = H(kj)$
      20: continue     /* now transform the four elements explicitly targeted by theta */
          $H(ii) = c * c * hii + s * s * hjj + two * c * s * hij$
          $H(jj) = c * c * hjj + s * s * hii - two * c * s * hij$
          $H(ij) = zero$;
          $H(ji) = zero$
    10: continue
      end do     /* Finish when largest off-diagonal is small enough */
  90: continue     /* Now sort the eigenvectors into eigenvalue order */
      $iq = -n$
      do $i = 1,\ n$
        $iq = iq + n$;
        $ii = loch(i,\ i)$;
        $jq = n * (i - 2)$
        do $j = i,\ n$
          $jq = jq + n$;
          $jj = loch(j,\ j)$
          if $(H(ii) < H(jj))$
            go to 30
          $temp = H(ii)$;
          $H(ii) = H(jj)$;
          $H(jj) = temp$
          do $k = 1,\ n$
            $ilr = iq + k$;
            $imr = jq + k$
            $temp = U(ilr)$;
            $U(ilr) = U(imr)$;
            $U(imr) = temp$
          end do
      30: continue
        end do
      end do
      return
    end
```

[matrix.web]

pack [`utilities.web`] Store the six electron repulsion labels.

`"main.f"` 6.10 ≡

```
subroutine pack(a, i, j, k, l, m, n)
  double precision a
  integer i, j, k, l, m, n

  double precision word
  integer id(6)
  character*1 chr1(8), chr2(24)
  equivalence (word, chr1(1)), (id(1), chr2(1))

  id(1) = i;
  id(2) = j;
  id(3) = k
  id(4) = l;
  id(5) = m;
  id(6) = n

  do ii = 1, 6
    chr1(ii) = chr2((ii - 1) * BYTES_PER_INTEGER + LEAST_BYTE)
  end do
  a = word
  return
end
```

[`utilities.web`]

unpack [`utilities.web`] Regenerate the 6 electron repulsion labels.

`"main.f"` 6.12 ≡

```
subroutine unpack(a, i, j, k, l, m, n)
   double precision a
   integer i, j, k, l, m, n

   double precision word
   integer id(6)
   character*1 chr1(8), chr2(24)
   equivalence (word, chr1(1)), (id(1), chr2(1))

   do ii = 1, 6
      chr2((ii - 1) * BYTES_PER_INTEGER + LEAST_BYTE) = chr1(ii)
   end do

   id(1) = i;
   id(2) = j;
   id(3) = k
   id(4) = l;
   id(5) = m;
   id(6) = n

   return
end
```

[`utilities.web`]

next_label [`utilities.web`] Generate the next label of electron repulsion integral.

A function to generate the four standard loops which are used to generate (or, more rarely) process the electron repulsion integrals.

The sets of integer values are generated in the usual standard order in canonical form, that is, equivalent to the set of loops:
do $i = 1,\ n$ { **do** $j = 1,\ i$ { **do** $k = 1,\ i$ { $ltop = k$ **if** $(i \equiv k)$ $ltop = j$ **do** $l = 1,\ ltop$ { **do** *somethingwithijkl* } } } }
Note that, just as is the case with the **do**-loops, the whole process must be *initialised* by setting initial values of i, j, k and l. If the whole set of labels is required then
$i = 1$, $j = 1$, $k = 1$, l=0
is appropriate.

Usage is, typically,
$i = 0$ $j = 0$ $k = 0$ $l = 0$
$while(next\_label(i,\ j,\ k,\ l,\ n) \equiv YES)$
{
do something with i j k and l
}

`"main.f"` 6.14 ≡

```
integer function next_label(i, j, k, l, n)
  integer i, j, k, l, n
  integer ltop
  next_label = YES
  ltop = k
  if (i ≡ k)
    ltop = j
  if (l < ltop) then
    l = l + 1
  else
    l = 1
    if (k < i) then
      k = k + 1
    else
      k = 1
      if (j < i) then
        j = j + 1
      else
        j = 1
        if (i < n) then
          i = i + 1
        else
          next_label = NO
        end if
      end if
    end if
  end if
  return
end
```

[utilities.web]

shalf [utilities.web] This subroutine calculates $\mathbf{S}^{-\frac{1}{2}}$ matrix from **S** matrix.

"main.f" 6.16 ≡

```
  subroutine shalf(S, U, W, m)
    implicit double precision (a − h, o − z)
    double precision S(∗), U(∗), W(∗)
    integer m

    data crit, one/1.0 · 10^−10 D, 1.0 · 10^+00 D/

    call eigen(S, U, m)     /∗ Transpose the eigenvalues of S for convenience ∗/
    do i = 1, m
      do j = 1, i
        ij = m ∗ (j − 1) + i;
        ji = m ∗ (i − 1) + j;
        d = U(ij)
        U(ij) = U(ji);
        U(ji) = d
      end do
    end do    /∗ Get the inverse root of the eigenvalues ∗/
    do i = 1, m
      ii = (i − 1) ∗ m + i
      if (S(ii) < crit) then
        write (ERROR_OUTPUT_UNIT, 200)
        STOP
      end if
      S(ii) = one / dsqrt(S(ii))
    end do
    call gtprd(U, S, W, m, m, m)
    call gmprd(W, U, S, m, m, m)

    return
200: format ("␣Basis␣is␣linearly␣deoendent;␣S␣is␣singular!␣")
  end
```

[utilities.web]

spinor [`utilities.web`]

`"main.f"` 6.18 ≡

```
subroutine spinor(H, m)
   double precision H(*)
   integer m

   double precision zero
   integer i, j, ij, ji, ip, jp, ijp, ijd, nl, n
   data  zero/0.0 · 10^{+00}D/

   n = 2 * m;
   nl = m + 1

   do i = 1, m
      do j = 1, m
         ij = m * (j − 1) + i;
         ip = i + m;
         jp = j + m
         ijp = n * (jp − 1) + ip;
         H(ijp) = H(ij)
      end do
   end do

   do i = 1, m
      do j = 1, m
         ip = i + m;
         jp = j + m;
         ijp = n * (jp − 1) + ip
         ijd = n * (j − 1) + i;
         H(ijd) = H(ijp)
      end do
   end do

   do i = 1, m
      do j = nl, n
         ij = n * (j − 1) + i;
         ji = n * (i − 1) + j
         H(ij) = zero
         H(ji) = zero
      end do
   end do

   return
end
```

[`utilities.web`]

7 INDEX

COMMAND LINE: `"fweave -C3 main.web"`.

WEB FILE: `"main.web"`.

CHANGE FILE: `(none)`.

GLOBAL LANGUAGE: FORTRAN.