December 6, 2014
19:14

Contents

getint This function withdraws (ij, kl) two-electron integral from the `file`.

```
/* STRUCTURES */
```

"gints.f" 0.1 ≡

```
@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

    /* OUTPUT STREAM UNITS */

@m ERROR_OUTPUT_UNIT 6
@m ERI_UNIT 17


  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
```

putint This function is just happy.

"gints.f" 0.3 ≡

```
  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
```

genint This subroutine generates one- and two-electron integrals.

"gints.f" 0.5 ≡

```
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 #6 ⟩

⟨ Normalize the primitives #7 ⟩

  /* 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

⟨ Copy GTO contraction coeffs to gtoC 0.6 ⟩ ≡

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

This code is used in section 0.5.

⟨ Normalize the primitives 0.7 ⟩ ≡

```
  /* First, normalize the primitives */
pitern = 5.568327997 · 10^{+00}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 / 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 / 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 0.5.

Utilities The utility functions

gtprd

[matrix.web]

"gints.f" 9 ≡
@m *loch*(*i*, *j*) (*n* ∗ (*j* − 1) + *i*)

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

  double precision zero
  integer k, ik, j, ir, ij, ib
  data zero/0.0 · 10^+00D/     /* 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]

"gints.f" 11 ≡

```
    subroutine gmprd(A, B, R, n, m, l)
      double precision A(MATRIX_SIZE), B(MATRIX_SIZE)
      double precision R(MATRIX_SIZE)
      integer n, m, l

      double precision zero
      integer k, ik, j, ir, ji, ib
      data zero/0.0 · 10^+00D/     /* 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]

"gints.f" 13 ≡

```
    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^+00D, 1.0 · 10^−20D, 1.0 · 10^+00D, 2.0 · 10^+00D,
          4.0 · 10^+00D, 1.0 · 10^+20D/    /* 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 90 while(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 ≡ k) | (j ≡ 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

Store the six electron repulsion labels.

"gints.f" 16 ≡

```
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
```

unpack

Regenerate the 6 electron repulsion labels.

"gints.f" 18 ≡

```
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
```

next_label

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** *something with ijkl* } } } }
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
}

"gints.f" 20 ≡

```
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
```

shalf

This subroutine calculates $\mathbf{S}^{-\frac{1}{2}}$ matrix from $\mathbf{S}$ matrix.

"gints.f" 22 ≡

```
  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^−10D, 1.0 · 10^+00D/
    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
```

⟨ Copy GTO contraction coeffs to gtoC 0.6 ⟩ Used in section 0.5.
⟨ Normalize the primitives 0.7 ⟩ Used in section 0.5.

COMMAND LINE: `"fweave -C3 gints.web"`.
WEB FILE: `"gints.web"`.
CHANGE FILE: `(none)`.
GLOBAL LANGUAGE: FORTRAN.