December 7, 2014
14:58

Contents

1 getint

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

```
"gints.f" 1 ≡
  @m ARB 1
  @m YES 0
  @m NO 1

  @m ERR −10
  @m OK 10

  @m BYTES_PER_INTEGER 4
  @m LEAST_BYTE 1

  @m END_OF_FILE −1

  @m NO_OF_TYPES 20
  @m INT_BLOCK_SIZE 20

  @m LAST_BLOCK 1
  @m NOT_LAST_BLOCK 0

  @m ERROR_OUTPUT_UNIT 6

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

  @m MAX_ITERATIONS 60

  @m UHF_CALCULATION 10
  @m CLOSED_SHELL_CALCULATION 20


    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
```

2 putint

This function is just happy.

"gints.f" 2 ≡

```
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
```

3 genint

This subroutine generates one- and two-electron integrals.

"gints.f" 3 ≡

```
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(MAX_PRIMITIVES)
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/
data gtoC/MAX_PRIMITIVES * 0.0 · 10^+00D/

mu = 0

⟨ Copy GTO contraction coeffs to gtoC 3.1 ⟩

⟨ Normalize the primitives 3.2 ⟩

  /* 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 3.1 ⟩ ≡

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

This code is used in section 3.

⟨ Normalize the primitives 3.2 ⟩ ≡

```
  /* 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 + D00
    t1 = dfact(l + 1) / alpha^l
    t2 = dfact(m + 1) / alpha^m
    t3 = dfact(n + 1) / alpha^n
    write(*, *) dfact(l + 1), dfact(m + 1), dfact(n + 1), l, m, 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 + 1) / t^l
      t2 = dfact(m + 1) / t^m
      t3 = dfact(n + 1) / t^n
      sum = sum + gtoC(ii) * gtoC(jj) * SOO * t1 * t2 * t3
    end do
  end do
  sum = one / dsqrt(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 3.

4 pack

Store the six electron repulsion labels.

"gints.f" 4 ≡

```
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
```

5 unpack

Regenerate the 6 electron repulsion labels.

"gints.f" 5 ≡

```
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
```

6 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" 6 ≡

```
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
```

7 INDEX

⟨ Copy GTO contraction coeffs to gtoC 3.1 ⟩ Used in section 3.
⟨ Normalize the primitives 3.2 ⟩ Used in section 3.

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