December 6, 2014
12:15

Contents

1 genoei

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*.

```
"integral.f" 1 ≡
  @m ERROR_OUTPUT_UNIT 6

    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 1.1 ⟩

      /∗ Insert the Factorials ∗/

    ⟨ Factorials 4.2 ⟩

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

    ⟨ One-electron Integer Setup 1.2 ⟩

      /∗ 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 4.4 ⟩     /∗ Use the Gaussian-product theorem to find $\vec{P}$ ∗/
        ⟨ Overlap Components 1.3 ⟩
        ovltot = ovltot + anorm ∗ bnorm ∗ ovl     /∗ accumulate Overlap ∗/
        ⟨ Kinetic Energy Components 2.1 ⟩
        kintot = kintot + anorm ∗ bnorm ∗ kin     /∗ accumulate Kinetic energy ∗/
```

```
            /* now the nuclear attraction integral */
          tnai = zero

          ⟨ Form fj 2.2 ⟩    /* 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 3.1 ⟩

            t = t1 * pcsq

            call auxg(m, t, g)    /* Generate all the $F_\nu$ required */

            ⟨ Form As 2.3 ⟩    /* 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 * vn, 4 )    /* Add to total 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
```

These are the declarations which are local to *genoei*, working space *etc.*

```
⟨ genoei local declarations 1.1 ⟩ ≡
      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/$0.0 \cdot 10^{00}$D, $1.0 \cdot 10^{00}$D, $2.0 \cdot 10^{00}$D, $0.5 \cdot 10^{00}$D, $0.25 \cdot 10^{00}$D/
      data pi/$3.141592653589 \cdot 10^{00}$D/
```

This code is used in section 1.

Get the various powers of x, y and z required from the data structures and obtain the contraction limits etc.

```
⟨ One-electron Integer Setup 1.2 ⟩ ≡
    $ityp = ntype(i)$;
    $jtyp = ntype(j)$
    $l1 = nr(ityp,\ 1)$;
    $m1 = nr(ityp,\ 2)$;
    $n1 = nr(ityp,\ 3)$
    $l2 = nr(jtyp,\ 1)$;
    $m2 = nr(jtyp,\ 2)$;
    $n2 = nr(jtyp,\ 3)$
    $imax = l1 + l2 + 1$;
    $jmax = m1 + m2 + 1$;
    $kmax = n1 + n2 + 1$
    $maxall = imax$
    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 1.

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 1.3 ⟩ ≡
    $prefa = two * prefa$
    $expab = \textbf{dexp}(-aexp * bexp * rAB / t1)$
    $s00 = (pi / t1)^{1.5} * expab$
    $dum = one;$
    $tf(1) = one;$
    $del = half / t1$
    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 1.

2 ovrlap

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.

"integral.f" 2 ≡

```
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 · 10^00D, 1.0 · 10^00D/
  if ((l1 < 0) | (l2 < 0)) then
    ovrlap = zero
    return
  end if
  if ((l1 ≡ 0) ∧ (l2 ≡ 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
```

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

⟨ Kinetic Energy Components 2.1 ⟩ ≡

```
    xl = dfloat(l2 ∗ (l2 − 1));
    xm = dfloat(m2 ∗ (m2 − 1))
    xn = dfloat(n2 ∗ (n2 − 1));
    xj = 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 1.

Form the f_j coefficients needed for the nuclear attraction integral.

```
⟨Form fj 2.2⟩ ≡
    $m = imax + jmax + kmax - 2$
    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 1.

Use *aform* to compute the required A-factors for each Cartesian component.

```
⟨Form As 2.3⟩ ≡
    $epsi = quart\ /\ t1$
    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 1.

3 aform

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:

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

"integral.f" 3 ≡

```
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 $one/1.0\cdot 10^{00}$D/

  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
```

Get the co-ordinates of the attracting nucleus with respect to $\vec{P}$.

⟨ Nuclear data 3.1 ⟩ ≡

```
$cpx=p(1)-vlist(n,\ 1)$
$cpy=p(2)-vlist(n,\ 2)$
$cpz=p(3)-vlist(n,\ 3)$
$pcsq=cpx*cpx+cpy*cpy+cpz*cpz$
```

This code is used in section 1.

4 generi

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.

"integral.f" 4 ≡

```
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 4.1 ⟩
    /* Insert the data statement for the factorials */
  ⟨ Factorials 4.2 ⟩
    /* 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 4.3 ⟩
    /* 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 4.4 ⟩
        /* 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 4.6 ⟩
      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 4.5 ⟩
```

```
            $w = pi \,/\, (t1 + t2)$

              /* 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 4.7 ⟩

            **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 4.8 ⟩

              /* 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 */

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

          **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**
```

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

⟨ generi local declarations 4.1 ⟩ ≡

```
    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.

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

⟨ Factorials 4.2 ⟩ ≡

```
    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 1, 4, and 5.

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 4.3 ⟩ ≡

```
    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.

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 4.4 ⟩ ≡
    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 = dexp(−aexp * bexp * rAB / t1) * pi * anorm * bnorm / t1
```

This code is used in sections 1 and 4.

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 4.5 ⟩ ≡

```
    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.

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 4.6 ⟩ ≡

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

    $mleft = i1max + j1max + k1max$

    **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.

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 4.7 ⟩ ≡

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

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

    **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.

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

```
⟨ Form Bs 4.8 ⟩ ≡
    $ppx(1) = one$;
    $bbx(1) = zero$
    $ppy(1) = one$;
    $bby(1) = zero$
    $ppz(1) = one$;
    $bbz(1) = zero$

    $jt1 = i1max + i2max$
    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.

5 fj

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.

"integral.f" 5 ≡

```
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 4.2 ⟩
  $imax = \textbf{min}(j, l)$
  $imin = \textbf{max}(0, j - m)$
  $sum = 0.0 \cdot 10^{00}\texttt{D}$
  do $i = imin, imax$
    $term = fact(l + 1) * fact(m + 1) / (fact(i + 1) * fact(j - i + 1))$
    $term = term / (fact(l - i + 1) * fact(m - j + i + 1))$
    $aa = 1.0 \cdot 10^{00}\texttt{D}$;
    $bb = 1.0 \cdot 10^{00}\texttt{D}$
    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
```

6 theta

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 $(ij,k\ell)$).

`"integral.f"` 6 ≡

```
    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/$0.0 \cdot 10^{00}$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
```

7 bform

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

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

"integral.f" 7 ≡

```
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
```

8 auxg

Find the maximum value of F_ν required, use *fmch* to compute it and obtain all the lower F_ν by downward recursion.

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

`"integral.f"` 8 ≡

```
  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
```

9 fmch

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 $x \leq 10$ (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.

"integral.f" 9 ≡

```
double precision function fmch(nu, x, y)
  ⟨Declarations 9.1⟩   /* First, make the variable declarations */
  ⟨Internal Declarations 9.2⟩
  $m = nu$
  $a = \mathbf{dfloat}(m)$
  if ($x \le ten$) then
    ⟨Small x Case 9.3⟩
  else
    ⟨Large x Case 9.4⟩
  end if
end
```

Here are the declarations and **data** statements which are ...

⟨ Declarations 9.1 ⟩ ≡

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

This code is used in section 9.

⟨ Internal Declarations 9.2 ⟩ ≡

```
    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 9.

⟨ Small x Case 9.3 ⟩ ≡

```
    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 9.

```
⟨ Large x Case 9.4 ⟩ ≡
    $b = a + half$
    $a = a - half$
    $xd = one \, / \, x$
    $approx = rootpi4 * \textbf{dsqrt}(xd) * xd^{m}$
    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 9.

10 INDEX

COMMAND LINE: `"fweave integral.web"`.

WEB FILE: `"integral.web"`.

CHANGE FILE: `(none)`.

GLOBAL LANGUAGE: Fortran.