December 5, 2014
1:44

Contents

1 GENPSE

Function to compute the one-electron integrals (overlap, kinetic energy and nuclear attraction plus pseudopotential). The STRUCTURES and GENERI 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*. The pseudopotential integrals are computed by the method of Komar.

"pseudor.f" 1 ≡
@m *ARB* 1
@m *YES* 0
@m *NO* 1
@m ERR −1

@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

double precision function *genpse*(*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*(*ngmx*, 5), *vlist*(*ncmx*, 4);

{

double precision *Airu*(10), *Ajsv*(10), *Aktw*(10);
double precision *p*(3), *sf*(10, 3), *tf*(20), *ca*(3), *ba*(3);
double precision *fact*(20), *g*(50);
double precision *kin*, *tnai*, *totnai*, *tpse*, *totpse*;
integer *bigZ*;

data *zero*, *one*, *two*, *half*, *quart*/0.0 · 10^{00}D, 1.0 · 10^{00}D, 2.0 · 10^{00}D, 0.5 · 10^{00}D, 0.25 · 10^{00}D/;

data $pi/3.141592653589 \cdot 10^{00}\mathtt{D}/$;

⟨ Factorials #2 ⟩⟨ One-electron Integer Setup #3 ⟩
/* Obtain the powers of x,y,z and summation limits */

$rAB = (eta(iss,\ 1) - eta(jss,\ 1))^2 + (eta(iss,\ 2) - eta(jss,\ 2))^2 + (eta(iss,\ 3) - eta(jss,\ 3))^2$;
/* Initialise all accumulators */

$genoei = zero$;
$totpse = zero$;
$totnai = zero$; $kintot = zero$; $ovltot = zero$;

for $(irun = iss;\ irun \le il;\ irun = irun + 1)$ /* start of "i" contraction */

{

for $(jrun = jss;\ jrun \le jl;\ jrun = jrun + 1)$ /* start of "j" contraction */

{

⟨ Compute PA #4 ⟩ /* Use the Gaussian-product theorem to find $\vec{P}$ */

⟨ Overlap Components #6 ⟩

$ovltot = ovltot + anorm * bnorm * ovl$; /* accumulate Overlap */

⟨ Kinetic Energy Components #7 ⟩

$kintot = kintot + anorm * bnorm * kin$; /* accumulate Kinetic energy */

/* now the nuclear attraction integral */
$tpse = zero$;
$tnai = zero$;

⟨ Form fj #8 ⟩ /* Generate the required f_j coefficients */

for $(n = 1;\ n \le noc;\ n = n + 1)$ /* loop over nuclei */

{

$bigZ = vlist(n,\ 4) + 0.001 \cdot 10^{00}\mathtt{D}$; /* round to integer */
/* Do pseudo potential first - remember vlist(n,4) is bigZ */
do $kpse = 1,\ 3$;

{

$ca(kpse) = eta(irun,\ kpse) - vlist(n,\ kpse)$; /* relative positions */
$ba(kpse) = eta(jrun,\ kpse) - vlist(n,\ kpse)$;

}

$pse = Vps(l1,\ m1,\ n1,\ aexp,\ ca,\ l2,\ m2,\ n2,\ bexp,\ ba,\ bigZ)$;

/* bigZ is changed to Zeff in Vps */

$tpse = tpse + pse$; /* pseudo potential added in - now for Z_{eff}/r nuclear attraction term */
$pn = zero$; /* Initialise current contribution */

/* Get the attracting-nucleus information; co-ordinates */

⟨ Nuclear data #11 ⟩

$t = t1 * pcsq$;

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

⟨ Form As #10 ⟩ /* Generate the geometrical A-factors */

```
  /* Now sum the products of the geometrical A-factors and the F_nu */
for (ii = 1; ii ≤ imax; ii = ii + 1)
  {
  for (jj = 1; jj ≤ jmax; jj = jj + 1)
    {
    for (kk = 1; kk ≤ kmax; kk = kk + 1)
      {
      nu = ii + jj + kk − 2;
      pn = pn + Airu(ii) * Ajsv(jj) * Aktw(kk) * g(nu);
      }
    }
  }
tnai = tnai − pn * float(bigZ);    /* Add to total multiplied by current charge */
}    /* end of loop over nuclei */
totnai = totnai + prefa * tnai;
totpse = totpse + anorm * bnorm * tpse;
}    /* end of "j" contraction */
}    /* end of "i" contraction */
genpse = totnai + totpse + kintot;    /* "T + V + Vpse" */
  /* write(6,200) i,j,ovltot,kintot,totnai,totpse,genpse; 200 format(2i3,5f12.5); */
return;
}
```

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

⟨ Factorials 1.1 ⟩ ≡

```
data fact/1.0, 1.0, 2.0, 6.0, 24.0, 120.0, 720.0, 5040.0, 40320.0, 362880.0, 3628800.0, 39916800.0,
    479001600.0, 6227020800.0, 6 * 0.0/;
```

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.

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 1.3 ⟩ ≡
    aexp = eta(irun, 4);
    anorm = eta(irun, 5);
    bexp = eta(jrun, 4);
    bnorm = eta(jrun, 5);
    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 = exp(-aexp * bexp * rAB / t1) * pi * anorm * bnorm / t1;
```

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 1.5 ⟩ ≡

```
prefa = two * prefa;
expab = exp(-aexp * bexp * rAB / t1);
s00 = (pi / t1)^1.5 * expab;
dum = one; tf(1) = one;
del = half / t1;
for (n = 2; n ≤ maxall; n = n + 1)
  {
  tf(n) = tf(n - 1) * dum * del; dum = dum + two; }
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.

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

⟨ Kinetic Energy Components 1.6 ⟩ ≡

```
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 1.7 ⟩ ≡
```
m = imax + jmax + kmax − 2;
for (n = 1; n ≤ imax; n = n + 1)
  sf(n, 1) = fj(l1, l2, n − 1, pax, pbx);
for (n = 1; n ≤ jmax; n = n + 1)
  sf(n, 2) = fj(m1, m2, n − 1, pay, pby);
for (n = 1; n ≤ kmax; n = n + 1)
  sf(n, 3) = fj(n1, n2, n − 1, paz, pbz);
```

This code is used in section 1.

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

⟨ Form As 1.9 ⟩ ≡
```
epsi = quart / t1;
for (ii = 1; ii ≤ 10; ii = ii + 1)
  {
  Airu(ii) = zero; Ajsv(ii) = zero; Aktw(ii) = zero;
  }
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.

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

⟨ Nuclear data 1.10 ⟩ ≡
```
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.

2 VPS

Function to organise the computation of the pseudo-potential integral for a particular choice of potential. This function has the duty of identifying the nature of the centre on which the source of effective potential is based (from *bigZ*) and getting the parameters for this atom from the arrays containing that information for many atoms. When this is done *Vps* calls *psepot* to do the actual integral evaluation.

"pseudor.f" 2 ≡

double precision function *Vps*(*l1*, *m1*, *n1*, *alpha*, *ca*, *l2*, *m2*, *n2*, *beta*, *ba*, *bigZ*);
implicit double precision (*a* − *h*, *o* − *z*);
integer *l1*, *l2*, *m1*, *m2*, *n1*, *n2*, *bigZ*; **double precision** *alpha*, *beta*, *ca*(∗), *ba*(∗) {
/∗ Pseudopotential arrays ∗/
integer *nupse*(*NU_DIM*);
integer *nc*(3), *nb*(3);
integer *kpsemx*, *lpsemx*; /∗ Current atom expansion maxima ∗/
integer *idparm*, *inparm*; /∗ Position of PS parameters ∗/
integer *doff*, *noff*; /∗ Offsets in PS arrays ∗/
double precision *dpse*(*DPSE_DIM*), *dzu*(*NU_DIM*);
double precision *pse*;
double precision *fourpi*;
double precision *psepot*; /∗ Pseudopotential function ∗/

data *fourpi*/12.56637061 · 10^{00}D/, *zero*/0.0 · 10^{00}D/; /∗ ¡PSE Data¿ has the parameters for the Pseudopotentials in the form of "data" statements for dpse, nupse, dzu ∗/

⟨ PSE Data #13 ⟩

nc(1) = *l1*; *nc*(2) = *m1*; *nc*(3) = *n1*;
nb(1) = *l2*; *nb*(2) = *m2*; *nb*(3) = *n2*;
doff = 1; /∗ 1st. Row parameters at start of dpse coefficient array ∗/
noff = 1; /∗ and at start of exponent and power arrays ∗/

/∗ Size of expansions for 1st.Row ∗/

kpsemx = *FIRST_ROW_KMAX*;
lpsemx = *FIRST_ROW_LMAX*;
if ((*NEON* < *bigZ*) ∧ (*bigZ* ≤ *ARGON*))

{

/∗ offsets for 2nd. row parameters ∗/

doff = *NUMBER_OF_FIRST_ROW* ∗ *FIRST_ROW_KMAX* ∗ (*FIRST_ROW_LMAX* + 1) + 1;
noff = *NUMBER_OF_FIRST_ROW* ∗ *FIRST_ROW_KMAX* + 1;

/∗ Size of expansions for 2nd. Row ∗/

kpsemx = *SECOND_ROW_KMAX*;
lpsemx = *SECOND_ROW_LMAX*;

}

if ((*ARGON* < *bigZ*) ∧ (*bigZ* ≤ *ZINC*))

{

/∗ offsets for 3rd. Row parameters including First Transition Series ∗/

```
    doff = NUMBER_OF_FIRST_ROW * FIRST_ROW_KMAX * (FIRST_ROW_LMAX + 1) +
        NUMBER_OF_SECOND_ROW * SECOND_ROW_KMAX * (SECOND_ROW_LMAX + 1) + 1;
    noff = NUMBER_OF_FIRST_ROW * FIRST_ROW_KMAX + NUMBER_OF_SECOND_ROW *
        SECOND_ROW_KMAX + 1;
      /* Size of expansions for 3rd. Row */
    kpsemx = THIRD_ROW_KMAX;
    lpsemx = THIRD_ROW_LMAX;
    }
  pse = zero;    /* Initialise the integral */
  if (bigZ ≤ HELIUM)    /* no potentials for H, He */
    return (pse);
  if ((HELIUM < bigZ) ∧ (bigZ ≤ NEON))
    bigZ = bigZ − HELIUM;
  if ((NEON < bigZ) ∧ (bigZ ≤ ARGON))
    bigZ = bigZ − NEON;
  if ((ARGON < bigZ) ∧ (bigZ ≤ ZINC))
    bigZ = bigZ − ARGON;
  if (bigZ > ZINC)
    {
    write (ERROR_OUTPUT_UNIT, 200) ;
    STOP;
    }
  200 format ("␣No␣Pseudo␣potential␣for␣this␣atom")
  idparm = (bigZ − 1) * kpsemx * (lpsemx + 1) + doff;
  inparm = (bigZ − 1) * kpsemx + noff;
  pse = psepot(nc, alpha, ca, nb, beta, ba, dzu(inparm), dpse(idparm), nupse(inparm), kpsemx,
      lpsemx);
  pse = pse * fourpi;
  return (pse);
}
    /* The (long and tedious) data structure containing the parameters for the effective potentials for
      the atoms included in the set. */
```

⟨ PSE Data 2.1 ⟩ ≡
@m *LDIM1P1* **$EVAL** (*LDIM1* + 1)
@m *LDIM1PLDIM2* **$EVAL** (*LDIM1* + *LDIM2*)
@m *LDIM1PLDIM2P1* **$EVAL** (*LDIM1PLDIM2* + 1)
data (*dpse*(*i*), *i* = 1, *LDIM1*) /3.48672 · 10^00D, 0.49988 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D,
−0.77469 · 10^00D, /∗ Li ∗/
0.99509 · 10^00D, −0.02612 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D, −0.27010 · 10^00D, /∗ Be ∗/
1.04649 · 10^00D, −0.07501 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D, −0.36886 · 10^00D, /∗ B ∗/
1.07785 · 10^00D, −0.17140 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D, −0.40843 · 10^00D, /∗ C ∗/
1.09851 · 10^00D, −0.33854 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D, −0.43676 · 10^00D, /∗ N ∗/
/∗ 1.11152d00, -0.60045d00, 0.0d00, 0.0d00, 0.0d00, -0.44108d00, O ∗/
1.6477 · 10^00D, 45.0783 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D, −7.7907 · 10^00D, /∗ O (new) ∗/
1.12060 · 10^00D, −0.98560 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D, −0.44625 · 10^00D, /∗ F ∗/
1.12861 · 10^00D, −1.55047 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D, −0.46631 · 10^00D/; /∗ Ne ∗/
data (*dpse*(*i*), *i* = *LDIM1P1*, *LDIM1PLDIM2*) /1.74854 · 10^00D, −0.01388 · 10^00D, 5 ∗ 0.0 · 10^00D,
1.46565 · 10^00D, 0.15319 · 10^00D, /∗ Na ∗/
5 ∗ 0.0 · 10^00D, −2.83231 · 10^00D, /∗ Na ∗/
2.00073 · 10^00D, −0.03023 · 10^00D, 5 ∗ 0.0 · 10^00D, 1.40926 · 10^00D, −0.00871 · 10^00D, /∗ Mg ∗/
5 ∗ 0.0 · 10^00D, −92.89133 · 10^00D, /∗ Mg ∗/
2.16121 · 10^00D, −0.06021 · 10^00D, 5 ∗ 0.0 · 10^00D, 1.40609 · 10^00D, −0.02270 · 10^00D, /∗ Al ∗/
5 ∗ 0.0 · 10^00D, −0.93152 · 10^00D, /∗ Al ∗/
2.30683 · 10^00D, −0.10463 · 10^00D, 5 ∗ 0.0 · 10^00D, 1.61465 · 10^00D, −0.04644 · 10^00D, /∗ Si ∗/
5 ∗ 0.0 · 10^00D, −0.18945 · 10^00D, 2.42266 · 10^00D, −0.16784 · 10^00D, 5 ∗ 0.0 · 10^00D, 1.72241 · 10^00D,
−0.08401 · 10^00D, /∗ P ∗/
5 ∗ 0.0 · 10^00D, −0.40202 · 10^00D, /∗ P ∗/
2.51686 · 10^00D, −0.25672 · 10^00D, 5 ∗ 0.0 · 10^00D, 1.79573 · 10^00D, −0.13150 · 10^00D, /∗ S ∗/
5 ∗ 0.0 · 10^00D, −0.74309 · 10^00D, /∗ S ∗/
2.60459 · 10^00D, −0.37281 · 10^00D, 5 ∗ 0.0 · 10^00D, 1.85329 · 10^00D, −0.20197 · 10^00D, /∗ Cl ∗/
5 ∗ 0.0 · 10^00D, −0.98109 · 10^00D, /∗ Cl ∗/
2.66818 · 10^00D, −0.52659 · 10^00D, 5 ∗ 0.0 · 10^00D, 1.90592 · 10^00D, −0.29464 · 10^00D, /∗ Ar ∗/
5 ∗ 0.0 · 10^00D, −1.35035 · 10^00D/; /∗ Ar ∗/
data (*dpse*(*i*), *i* = *LDIM1PLDIM2P1*, *DPSE_DIM*) /135 ∗ 0.0 · 10^00D, /∗ Ditto for K to Co ∗/
5.57207 · 10^00D, −0.11685 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D,
5.49578 · 10^00D, 0.23560 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D,
−6.10669 · 10^00D, /∗ Ni ∗/
5.75316 · 10^00D, −0.13096 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D,
5.98399 · 10^00D, 0.55456 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D, 0.0 · 10^00D,
−6.12527 · 10^00D, /∗ Cu ∗/
15 ∗ 0.0 · 10^00D/; /∗ Fill in for Zn ∗/
data *dzu*/1.04883 · 10^00D, 1.04883 · 10^00D, 1.40580 · 10^00D, /∗ Li ∗/
0.25784 · 10^00D, 0.25784 · 10^00D, 0.96124 · 10^00D, /∗ Be ∗/
0.40255 · 10^00D, 0.40255 · 10^00D, 1.91861 · 10^00D, /∗ B ∗/
0.57656 · 10^00D, 0.57656 · 10^00D, 3.18301 · 10^00D, /∗ C ∗/
0.77911 · 10^00D, 0.77911 · 10^00D, 4.76414 · 10^00D, /∗ N ∗/
/∗ 1.00639d00, 1.00639d00, 6.34314d00, O ∗/
10.37387 · 10^00D, 10.37387 · 10^00D, 25.320084, /∗ O(new) ∗/
1.26132 · 10^00D, 1.26132 · 10^00D, 8.17605 · 10^00D, /∗ F ∗/
1.53611 · 10^00D, 1.53611 · 10^00D, 10.74498 · 10^00D, /∗ Ne ∗/
2 ∗ 0.25753 · 10^00D, 2 ∗ 0.50138 · 10^00D, 1.01908 · 10^00D, /∗ Na ∗/
2 ∗ 0.30666 · 10^00D, 2 ∗ 0.27481 · 10^00D, 6.35656 · 10^00D, /∗ Mg ∗/
2 ∗ 34298 · 10^00D, 2 ∗ 0.28766 · 10^00D, 0.85476 · 10^00D, /∗ Al ∗/

```
    0.39512 · 10^00D, 0.39512 · 10^00D, 0.40442 · 10^00D, 0.40442 · 10^00D, 0.25050 · 10^00D,    /* Si */
    0.45424 · 10^00D, 0.45424 · 10^00D, 0.49582 · 10^00D, 0.49582 · 10^00D, 0.56256 · 10^00D,    /* P */
    0.51644 · 10^00D, 0.51644 · 10^00D, 0.59819 · 10^00D, 0.59819 · 10^00D, 1.13649 · 10^00D,    /* S */
    0.59299 · 10^00D, 0.59299 · 10^00D, 0.69783 · 10^00D, 0.69783 · 10^00D, 2.00000 · 10^00D,    /* Cl */
    0.66212 · 10^00D, 0.66212 · 10^00D, 0.80903 · 10^00D, 0.80903 · 10^00D, 3.50000 · 10^00D,    /* Ar */
    45 * 0.0 · 10^00D,    /* Ditto for K to Co */
    0.63800 · 10^00D, 0.638000 · 10^00D, 0.70978 · 10^00D, 0.70978 · 10^00D, 2.44040 · 10^00D,    /* Ni */
    0.71270 · 10^00D, 0.71270 · 10^00D, 0.89496 · 10^00D, 0.89498 · 10^00D, 2.67294 · 10^00D,    /* Cu */
    5 * 0.0 · 10^00D/;    /* Fill in Zn later */
  data nupse/    /* these are nu + 2 values */
    0, 4, 1,    /* Li */
    0, 4, 1,    /* Be */
    0, 4, 1,    /* B */
    0, 4, 1,    /* C */
    0, 4, 1,    /* N */
     /* 0, 4, 1, O */
    1, 2, 2,    /* O (new) */
    0, 4, 1,    /* F */
    0, 4, 1,    /* Ne */
    0, 4, 0, 4, 1,    /* Na */
    0, 4, 0, 4, 1,    /* Mg */
    0, 4, 0, 4, 1,    /* Al */
    0, 4, 0, 4, 1,    /* Si */
    0, 4, 0, 4, 1,    /* P */
    0, 4, 0, 4, 1,    /* S */
    0, 4, 0, 4, 1,    /* Cl */
    0, 4, 0, 4, 1,    /* Ar */
    0, 4, 0, 4, 1,    /* K */
    0, 4, 0, 4, 1,    /* Ca */
    0, 4, 0, 4, 1,    /* Sc */
    0, 4, 0, 4, 1,    /* Ti */
    0, 4, 0, 4, 1,    /* V */
    0, 4, 0, 4, 1,    /* Cr */
    0, 4, 0, 4, 1,    /* Mn */
    0, 4, 0, 4, 1,    /* Fe */
    0, 4, 0, 4, 1,    /* Co */
    0, 4, 0, 4, 1,    /* Ni */
    0, 4, 0, 4, 1,    /* Cu */
    0, 4, 0, 4, 1/;    /* Zn */
```

@Lr:

This code is used in section 2.

3 PSEPOT

Function to do the actual work of pseudopotential integral evaluation. The function calculates matrix elements of atomic pseudo- potential Vps centred on site A between two Gaussians centred on sites B and C (all three sites may assume arbitrary positions in space; any two of them or all three are allowed to coincide).

The coding here is "developed" (read that "mostly copied") from the FORTRAN 66 program described by M. Kolar (Comp. Phys. Communications **23**, 275 (1980) whose program description is a model of clear information. */

"pseudor.f" 3 ≡

double precision function *psepot*(*nc*, *zc*, *ca*, *nb*, *zb*, *ba*, *zu*, *Td*, *nu*, *kmax*, *lmax*);
implicit double precision (*a* − *h*, *o* − *z*);
integer *nc*(∗), *nb*(∗), *nu*(∗), *kmax*, *lmax*;
double precision *zc*, *zb*, *ca*(∗), *ba*(∗), *zu*(∗), *Td*(∗); { **double precision** *r1*, *r11*, *r*;
/∗ These three are entry points to *rinit* ∗/
integer *rinit*, *rl*, *rl1*; /∗ These three are the integer entry points to the same routine: *rinit* ∗/

⟨ psepot Declarations, commons and equivalences #15 ⟩

⟨ Use capi to get the geometric factors #16 ⟩

tzcb= *zc* + *zb*;
ncb = *ncn* + *nbn*; **if** (*logc* ∧ *logb*)

{ ⟨ Simple case, both distances zero #17 ⟩}

log = *logc* ∨ *logb*;
td1 = *two* ∗ *tca* ∗ *zc*; *td2* = *two* ∗ *tba* ∗ *zb*; *t2* = *zc* ∗ *zb*;
tg1 = *t2* ∗ (*tba* − *tca*)^2; *tg2* = *zb* ∗ *tba2* + *zc* ∗ *tca2*;
a(1) = *td1* + *td2*; **if** (***log***)

{ ⟨ Intermediate case, only one distance zero #18 ⟩}

⟨ General case, both distances non-zero #19 ⟩}

Here are the declaractions for work-space and intermediate storage.

⟨ psepot Declarations, commons and equivalences 3.1 ⟩ ≡
logical ***log***, *logc*, *logb*, *lgo*, *lgl*(252), *lgl1*(233);
double precision *tpic*(9, 4, 6), *tpib*(9, 4, 6), *tpc*(9), *tpb*(189), *ttpp*(189);
double precision *a*(2);
double precision *zero*, *quarter*, *half*, *one*, *two*;
common/cocapi/ *lgo*;
common/in1/ *lc*, *lc1*, *lc2*;
common/c0/ *tzcb*, *td1*, *td2*, *tg1*, *tg2*, *tg3*, *a*, *ts*, *ncb*, *jj*, *imax*, ***log***;
equivalence (*tpc*(1), *tpic*(1, 1, 1), *ttpp*(40)), (*tpb*(1), *tpib*(1, 1, 1), *t0*);
data *zero*/0.0 · 10^{00}D/, *one*/1.0 · 10^{00}D/, *two*/2.0 · 10^{00}D/;
data *quarter*, *half*/0.25 · 10^{00}D, 0.5 · 10^{00}D/;

This code is used in section 3.

Set up the geometrical factors with *capi*

```
⟨ Use capi to get the geometric factors 3.2 ⟩ ≡
    lc = lmax + 1; lc1 = lc + 1; lc2 = lc * lc;
    call capi(nc, ncn, ca, tca, tca2, lamx1, tpic, logc);
    if (¬lgo)
      return (zero);
    if (logc)
      {
      for (lm = 1; lm ≤ lc2; lm = lm + 1)
        tpb(lm) = tpc(lm);
      call capi(nb, nbn, ba, tba, tba2, lamx1, tpic, logb);
      }
    else
      call capi(nb, nbn, ba, tba, tba2, la1mx1, tpib, logb);
    if (¬lgo)
      return (zero);
```

This code is used in section 3.

This code is used when both basis functions are on the same centre as the core potential.

```
⟨ Simple case, both distances zero 3.3 ⟩ ≡
      /* Both distances tca and tcb are zero */
    t0 = t0 * tpc(1);
    if (lc > 1)
      {
      lm2 = 1;
      for (l = 2; l ≤ lc; l = l + 1)
        {
        lm1 = lm2 + 1;
        lm2 = l * l;
        t2 = zero;
        for (lm = lm1; lm ≤ lm2; lm = lm + 1)
          t2 = t2 + tpc(lm) * tpb(lm);
        tpb(l) = t2;
        }
      }
    t2 = zero;
    for (k = 1; k ≤ kmax; k = k + 1)
      {
      t1 = zero;
      for (l = 1; l ≤ lc; l = l + 1)
        t1 = t1 + tpb(l) * td((l − 1) * kmax + k);
      if (t1 ≡ zero)
        next;
      t2 = t2 + r11(zu(k), nu(k)) * t1;
      }
    psepot = t2;
    return;
```

This code is used in section 3.

Only one of the basis functions is on a centre different from the core-potential centre.

⟨ Intermediate case, only one distance zero 3.4 ⟩ ≡

```
/* One and only one of both distances tca, tba is nonzero, swap the parameters if it is the wrong
   one */
if (¬logb)
  {
  n1 = ncn;
  ncn = nbn;
  nbn = n1;
  }
jj = 1;
imax = lamx1 + ncn;
nc1 = ncn + 1;
i = 1; ii = 1; iii = 0;
for (la = 1; la ≤ lamx1; la = la + 1)
  {
  l1 = max0(1, la − ncn);
  for (ka = 1; ka ≤ nc1; ka = ka + 1)
    {
    lm2 = (l1 − 1)^2;
    lgl(i) = T;
    irl = lc1 − l1;
    for (l = l1; l ≤ lc; l = l + 1)
      {
      lm1 = lm2 + 1;
      lm2 = l ∗ l;
      t1 = zero;
      for (lm = lm1; lm ≤ lm2; lm = lm + 1)
        t1 = t1 + tpb(lm) ∗ tpic(lm, ka, la);
      lgl1(ii) = t1 ≡ zero;
      if (lgl1(ii))
        {
        ii = ii + 1; next;
        }
      lgl(i) = F;
      iii = iii + 1;
      ttpp(iii) = t1;
      ii = ii + 1;
      }
    if (lgl(i))
      ii = ii − irl;
    i = i + 1;
    }
  }
if (iii ≡ 0)
  return (zero);
t0 = zero;
for (k = 1; k ≤ kmax; k = k + 1)
  {
  nu1 = rinit(zu(k), nu(k)) + nbn;
  i = 1; ii = 1; iii = 0;
```

```
for (la = 1; la ≤ lamx1; la = la + 1)
  {
  l1 = max0(1, la − ncn);
  irl = rl1(la);    /* irl=rl1(la,isilly); */
  t1 = zero;
  for (ka = 1; ka ≤ nc1; ka = ka + 1)
    {
    if (lgl(i))
      {
      i = i + 1; next;
      }
    t2 = zero;
    for (l = l1; l ≤ lc; l = l + 1)
      {
      if (lgl1(ii))
        {
        ii = ii + 1; next;
        }
      iii = iii + 1;
      t2 = t2 + td((l − 1) ∗ kmax + k) ∗ ttpp(iii);
      ii = ii + 1;
      }
    if (t2 ≡ zero)
      {
      i = i + 1; next;
      }    /* t1=t1+t2*r1(nu1+ka,isilly); */
    t1 = t1 + t2 ∗ r1(nu1 + ka, la);
    i = i + 1;
    }
  t0 = t0 + t1 ∗ irl;
  }
}
psepot = t0 ∗ half;
return (psepot);
```

This code is used in section 3.

The general case, both basis functions are on centres which are different from the core-potential centre.

```
⟨ General case, both distances non-zero 3.5 ⟩ ≡
    /* Both the distances tca and tcb are not zero */
  tg3 = t2 * (tba + tca)^2; t1 = td1 - td2;
  ts = dsign(one, t1);
  a(2) = ts * t1;
  imax = 2 * (ncb + lc) - 1; jj = 2; nmax = ncb + 1;
  nc1 = ncn + 1; i = 1; ii = 1; iii = 0;
  for (la = 1; la ≤ lamx1; la = la + 1)
    {
    l2 = max0(1, la - ncn);
    for (la1 = 1; la1 ≤ la1mx1; la1 = la1 + 1)
      {
      l1 = max0(l2, la1 - nbn);
      irl = lc1 - l1;
      for (n = 1; n ≤ nmax; n = n + 1)
        {
        n1 = n + 1;
        kamin = max0(1, n - nbn);
        kamax = min0(nc1, n);
        lgl(i) = T;
        lm2 = (l1 - 1)^2;
        for (l = l1; l ≤ lc; l = l + 1)
          {
          lm1 = lm2 + 1; lm2 = l * l;
          t1 = zero;
          for (lm = lm1; lm ≤ lm2; lm = lm + 1)
            {
            for (ka = kamin; ka ≤ kamax; ka = ka + 1)
              {
              n1ka = n1 - ka;
              t1 = t1 + tpic(lm, ka, la) * tpib(lm, n1ka, la1);
              }
            }
          lgl1(ii) = t1 ≡ zero;
          if (lgl1(ii))
            {
            ii = ii + 1; next;
            }
          lgl(i) = F;
          iii = iii + 1;
          ttpp(iii) = t1;
          ii = ii + 1;
          }
        if (lgl(i))
          ii = ii - irl;
        i = i + 1;
        }
      }
    }
  if (iii ≡ 0)
```

```
    return (zero);
  for (i = 1; i ≤ iii; i = i + 1)
    tpb(i) = zero;
  for (k = 1; k ≤ kmax; k = k + 1)
    {
    nu1 = rinit(zu(k), nu(k));
    i = 1; ii = 1; iii = 0;
    for (la = 1; la ≤ lamx1; la = la + 1)
      {
      l2 = max0(1, la − ncn);
      for (la1 = 1; la1 ≤ la1mx1; la1 = la1 + 1)
        {
        l1 = max0(l2, la1 − nbn);
        irl = rl(la, la1);
        for (n = 1; n ≤ nmax; n = n + 1)
          {
          if (lgl(i))
            {
            i = i + 1; next;
            }
          t1 = r(nu1 + n) ∗ irl;
          for (l = l1; l ≤ lc; l = l + 1)
            {
            if (lgl1(ii))
              {
              ii = ii + 1; next;
              }
            iii = iii + 1;
            tpb(iii) = tpb(iii) + t1 ∗ td((l − 1) ∗ kmax + k);
            ii = ii + 1;
            }
          i = i + 1;
          }
        }
      }
    }
  t1 = zero;
  for (i = 1; i ≤ iii; i = i + 1)
    {
    t1 = t1 + ttpp(i) ∗ tpb(i);
    }
  psepot = t1 ∗ quarter;
  return (psepot);
```

This code is used in section 3.

4 derf

This is just the simplest polynomial taken from Abramowitz and Stegun. It needs checking for accuracy.

"pseudor.f" 4 ≡

```
double precision function derf(x);
implicit double precision (a-h, o-z);
double precision x;

{
data one/1.0d00/, p/0.3275911d00/;
data a1, a2, a3, a4, a5/0.254829592, -0.284496736d00, 1.421413741d00,
     -1.453152027d00, 1.061405429d00/;
xx = dabs(x);
t = one / (one + p * xx);
ee = dexp(-xx * xx);
derf = one - (a1 * t + a2 * t * t + a3 * t * t * t + a4 * t * t * t * t + a5 * t * t * t * t * t) * ee;
return;
}
```

⟨ Compute PA 1.3 ⟩ Used in section 1.
⟨ Factorials 1.1 ⟩ Used in section 1.
⟨ Form As 1.9 ⟩ Used in section 1.
⟨ Form fj 1.7 ⟩ Used in section 1.
⟨ General case, both distances non-zero 3.5 ⟩ Used in section 3.
⟨ Intermediate case, only one distance zero 3.4 ⟩ Used in section 3.
⟨ Kinetic Energy Components 1.6 ⟩ Used in section 1.
⟨ Nuclear data 1.10 ⟩ Used in section 1.
⟨ One-electron Integer Setup 1.2 ⟩ Used in section 1.
⟨ Overlap Components 1.5 ⟩ Used in section 1.
⟨ PSE Data 2.1 ⟩ Used in section 2.
⟨ Simple case, both distances zero 3.3 ⟩ Used in section 3.
⟨ Use capi to get the geometric factors 3.2 ⟩ Used in section 3.
⟨ psepot Declarations, commons and equivalences 3.1 ⟩ Used in section 3.

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