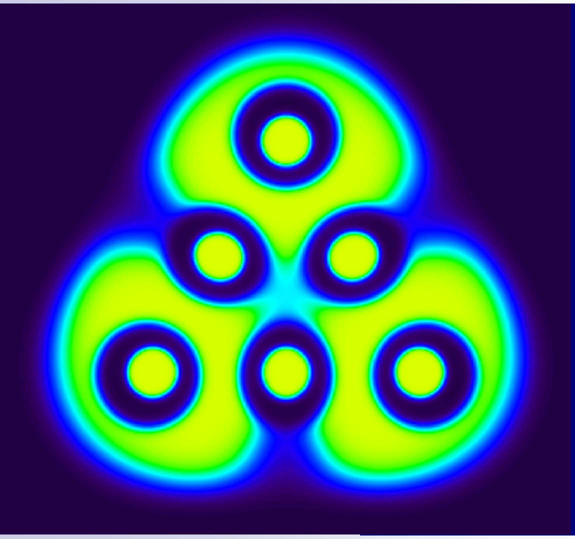


Modifying source code of Multiwfn

Tian Lu
sobereva@sina.com

Beijing Kein Research Center for Natural Sciences

http: //www.keinsci.com

Last update: 2024-Mar-25

Preface

The code of Multiwfn is easy to read, modify and extend. The framework of Multiwfn provides a flexible and convenient framework to develope new functions.

If you have any question about modifying or extending Multiwfn code, please feel free to post on Multiwfn forum or send question to E-mail of developer (Tian Lu).

PS: When publishing articles using the modified version of Multiwfn, the original paper of Multiwfn should be properly cited.

1

Structure, variables and functions of Multiwfn

Convention of Multiwfn code

- Multiwfn is written based on Fortran 95 (except for additionally invoked libraries), and only uses features of later versions of Fortran in very few places
- Multiwfn stores data in atomic unit internally
- Most of the code uses double precision float (8 bytes) and long integer (4 bytes) variables
- Implicit declaration is used for almost all subroutines via *implicit real*8 (a-h,o-z)*
- In the case where an array may be large, dynamic array should be used, so that its size is allocatable according to the actual situation. This avoids the upper limit of the analyzable size and the waste of memory
- Any time-consuming code should be parallelized with OpenMP technique. Multiplication between very large matrices should use function "matmul_blas" in *util.f90* for efficient evaluation.

Source files of Multiwfn

(1) Files shared by various functions

- **Multiwfn.f90**: Entry of Multiwfn. Initialize some global variables, load input file, and show main menu
- **define.f90** (module defvar; topo; surfvertex; basinintmod; NAOmod): Define modules which record global variables. All subroutines or functions that analyze the loaded system at least need to use defvar module (namely adding *use defvar*)
- **sub.f90**: Subroutines for realizing miscellaneous purposes, global variables are used. Also includes subroutines in main function 6 (modifying wavefunction)
- **util.f90** (module util): Including various utility subroutines for math, geometry, vector, matrix, string, etc., which are frequently employed by various functions of Multiwfn. Global variables are not involved.

- **GUI.f90** (module GUI): Subroutines invoking DISLIN library for creating GUI
- **plot.f90** (module plot): Subroutines invoking DISLIN library for plotting
- **function.f90** (module functions): Subroutines for evaluating various real space functions
- **fileIO.f90**: Subroutines of loading and exporting various kinds of files
- integral.f90: Generating electronic integrals
- NAONBO.f90: Subroutines related to NAO and NBO
- PBC.f90: Subroutines related to periodic boundary condition
- grid.f90: Subroutines related to grid data

- atmraddens.f90: Containing prefitted atomic radial densities
- edflib.f90: EDF library provided by Wenli Zou
- DFTxclib.F: Exchange-correlation functional library
- Bspline.f90: Library of performing one~six dimension B-spline interpolation
- minpack.f90: Library for non-linear fitting
- libreta.f90 and files in "libreta" folder: Code for evaluating electrostatic potential based on LIBRETA electron integral library
- fparser.f90: Library of translating string to mathematical expression
- O1.f90: Some codes which are compiled using -O1 option under Linux (unlike others, which are compiled using -O2). This is because due to possible bug of ifort compiler, some codes do not work properly if compiled with -O2 under Linux

(2) Files corresponding to various main functions

- 0123dim.f90: Study properties of a point, calculate and plot data for a line, a plane, and 3D spatial region (Main functions 1, 3, 4, 5)
- topology.f90: Topology analysis (Main function 2)
- population.f90: Population analysis (Main function 7)
- orbcomp.f90: Orbital composition analysis (Main function 8)
- bondorder.f90: Bond order analysis (Main function 9)
- DOS: Plotting DOS (Main function 10)
- spectrum.f90: Plotting various spectra (Main function 11)

- surfana.f90: Quantitative molecular surface analysis (Main function 12)
- procgriddata.f90: Process grid data (Main function 13)
- AdNDP.f90: AdNDP analysis (Main function 14)
- fuzzy.f90: Fuzzy atomic space analysis (Main function 15)
- CDA.f90: CDA analysis (Main function 16)
- basin.f90: Basin analysis (Main function 17)
- excittrans.f90: Electronic excitation anslysis (Main function 18)
- orbloc.f90: Orbital localization (Main function 19)

- visweak.f90: Visual analysis of weak interactions (Main function 20)
- EDA.f90: Energy decomposition analysis (Main function 21)
- CDFT.f90: Conceptual density functional analysis (Main function 22)
- ETS_NOCV.f90: ETS-NOCV analysis (Main function 23)
- hyper_polar.f90: (Hyper)polarizability analysis (Main function 24)
- deloc_aromat.f90: Delocalization and aromaticity analyses (Main function 25)
- cp2kmate.f90: Various auxiliary tools for CP2K program
- otherfunc/otherfunc2/otherfunc3.f90: Other functions, part 1/2/3 (Main functions 100/200/300)

Important global variables

Defined by module "defvar" in *define.f90*

See comments in this file for more information

Unit conversion

Bohr $\rightarrow$ Angstrom: b2a=0.529177249

Hartree $\rightarrow$ kcal/mol: au2kcal=627.51

Hartree $\rightarrow$ kJ/mol: au2KJ=2625.5

Hartree $\rightarrow$ eV: au2eV=27.2113838

a.u. $\rightarrow$ Debye: au2debye=2.5417462

amu $\rightarrow$ kg: amu2kg=1.66053878D-27

Physical constants

π: pi=3.141592653589793

Electron mass (kg): masse=9.10938215D-31

Light speed (m/s): lightc=2.99792458D8

Planck constant (J·s): planckc=6.62606896D-34

Reduced Planck constant (J·s): h_bar=1.054571628D-34

Boltzmann constant (J/K): boltzc=1.3806503D-23

Boltzmann constant (Hartree/K): boltzcau=3.1668114D-6

Boltzmann constant (eV/K): boltzceV=8.6173324D-5

Avogadro constant: avogacst=6.02214179D23

Information related to elements

Radii will be converted to Bohr as unit when Multiwfn boots up.

Element information starts from 0 (corresponding to ghost atoms) and ends at 150

- **covr**: CSD radii, *Dalton Trans.*, **2008**, 2832
- **covr_Suresh**: Suresh radii, *JPCA*, **105**, 5940
- **covr_pyy**: Pyykko radii, *Chem. Eur. J.*, **15**, 186
- **covr_tianlu**: The same as CSD, but use radius of group IVA of the same row
- **radii_Hugo**: Hugo radii, *CPL*, **480**, 127

  - **vdwr**: Bondi vdW radii, *JPC*, **68**, 441
  - **vdwr_tianlu**: The same as Bondi, but use radius of group IVA of the same row
  - **vdwr_UFF**: UFF vdW radii, *JACS*, **114**, 10024. Defined for first 103 elements

- **ind2name**: Converting element index to element name. First character uppercase, second lowercase
- **ind2name_up**: The same as above, but all uppercase characters
- **atmwei**: Abundance-averaged element mass

One-dimension grid data

- **curvex**: Store X value of curve data
- **curvey**: Store Y value of curve data
- **curveytmp**: The same as curvey, but temporarily store another set of Y data

Two-dimension grid data

- **planemat**: Store plane data for plotting plane map
- **planemattmp**: Store another set of plane data, grid setting is the same as planemat
- **orgx2D, orgy2D, orgz2D**: X/Y/Z of origin of plane data
- **ngridnum1, ngridnum2**: Number of points of plane data in two directions
- **v1x, v1y, v2x, v2y, v1z, v2z**: X/Y/Z of the two translation vectors of plane data
- **d1, d2**: Lengths of the two translation vectors

Three-dimension grid data

- **cubmat**: Three dimensional array storing 3D grid data
- **cubmattmp**: Temporarily storing another 3D grid data, with the same grid definition as cubmat
- **cubmatvec**: Storing 3D vector field data, with the same grid definition as cubmat
- **nx, ny, nz**: Number of grid points of cubmat in three dimensions (the grid is not necessarily orthogonal)
- **dx, dy, dz**: Grid spacing of cubmat in X/Y/Z
- **gridv1**(:), **gridv2**(:), **gridv3**(:): Three translation vectors of cubmat
- **orgx, orgy, orgz**: Origin of cubmat
- **endx, endy, endz**: End point of cubmat

NOTE: For orthogonal box, dx=gridvec1(1), dy=gridvec2(2), z=gridvec3(3)

Atom information

- **ncenter**: Total number of atoms in current system
- **a(*i*)%name**: name of atom *i*
- **a(*i*)%x, %y, %z**: X, Y, Z of atom *i* in Bohr
- **a(*i*)%index**: Element index of atom *i*
- **a(*i*)%charge**: Effective nuclear charge of atom *i*, it can also be the atomic charge loaded from .chg file. In the case of using pseudopotential, the difference between "index" and "charge" properties corresponds to the number of core electrons represented by pseudopotential
- **a(*i*)%resid**: Residue index of atom *i*
- **a(*i*)%resname**: Residue name of atom *i*

NOTE: "resid" and "resname" are defined only when pdb/pqr/gro file is used as input, and in this case global variable "iresinfo" will be 1, else 0 ("resid" is automatically set to 1, and "resname" is blank)

- **connmat**: Connectivity matrix between atoms, its (*i*, *j*) element records formal bond order. It can either be generated by subroutine "genconnmat", or generated when loading .mol/mol2 file, or loaded from .mol file via subroutine "readmolconn"
- **fragatm**: Storing fragment consisting of atoms, with "nfragatm" members. Fragment is used by some functions of Multiwfn
- **fragatm_org**: The same as above, but for backup purpose

Wavefunction information

- **wfntype**: Wavefunction type

0/1/2 = R/U/RO- single-determinant methods (*e.g.* HF and KS-DFT). More specifically, any case that all orbitals have integer occupation

3/4 = R/U- multiconfiguration methods (*e.g.* post-HF), the orbitals should be natural orbitals of corresponding level. More specificially, any case that some orbitals have non-integer occupation (*e.g.* NBO, NAO, NTO)

- **nelec, naelec, nbelec**: Number of total, α and β electrons
- **ninnerelec**: Number of core electrons represented by EDF
- **nmo**: Number of orbitals
- **MOocc(*i*)**: Occupation of orbital *i*
- **MOene(*i*)**: Energy of orbital *i* in Hartree
- **MOsym(*i*)**: Irreducible representation of orbital *i*
- **MOtype(*i*)**: Type of orbital *i*

  1 = α+β (closed-shell orbital), 2 = α, 3 = β
- **imodwfn**: Labelling if orbital information has been changed (*e.g.* via some options in main function 6). 0 by default (unchanged)
- **idxHOMO**: Index of HOMO, can be determined by subroutine "getHOMOidx"

Information related to GTF (Gaussian type function)

Cartesian form of GTF is always used

- **b(*i*)%center**: Index of atom that GTF *i* belongs to
- **b(*i*)%type**: Type of GTF *i* (*e.g.* X, YZZ). See "GTFtype2name" array in module "defvar" for meaning of the index
- **b(*i*)%exp**: Exponent of GTF *i*
- **nprims**: Number of GTFs
- **nprimshell**: Number of GTF shells
- **primshexp(*i*), primshcoeff(*i*)**: Exponent and contraction coefficient of GTF shell *i*
- **primstart(*i*), primend(*i*)**: Starting and ending GTF indices corresponding to basis function *i* (Cartesian type)
- **primconnorm(*j*)**: Normalization factor multiplied to contraction coefficient of GTF *i*

Coefficient matrices of GTFs

- **CO(*i*, *j*)**: Coefficient of GTF *j* in orbital *i*, both contraction coefficients and normalization factors have been included.

For unrestricted wavefunction, α and β oribtals are recorded in the first and second halves, respectively.

Dimension is (nmo,nprims).

Unique GTFs

When using a general contracted basis set, there will be some duplicated GTFs, namely some of them share same center, type and exponent.

To reduce cost during evaluating GTF contributions to real space function space, unique GTFs are automatically determined after loading a wavefunction file by subroutine "gen_GTFuniq". In subroutines "orbderv" and "orbderv_PBC", only unique GTFs are taken into account.

Related variables and arrays:

- nprims_GTF: Number of unique GTFs
- b_uniq: b array of unique GTFs
- CO_uniq: CO array of unique GTFs

Various matrices represented by GTFs

- **Ptot_prim, Palpha_prim, Pbeta_prim**: Total, α, and β density matrices, with dimension of (nprims,nprims). For restricted wavefunction only Ptot is defined. Can be generated by subroutine "gen_prim"
- **Dprim**: Electric dipole moment integral matrix in the basis of GTF, can be generated by subroutine "genDprim"
- **Quadprim, Octoprim, Hexdeprim**: Electric quadrupole, octopole and hexadecapole moment integral matrices in the basis of GTF, can be generated by subroutine "genMultipoleprim"

Information related to basis function

- **nbasis, nbasisCar**: Number of basis functions, number of Cartesian basis functions
- **nindbasis**: Number of linearly independent basis functions, can be loaded from mwfn and fch files. For other files, it is 0 (undefined)
- **nshell**: Number of basis function shells
- **shtype(*i*), shcon(*i*), shcen(*i*)**: Type, contraction degree and corresponding atom of basis function shell *i*. **shtypeCar(:)** is the shell type when all basis functions are Cartesian form
- **basshell(*i*)**: Shell that basis function *i* belongs to
- **bascen(*i*)**: Atom that basis function *i* belongs to
- **bastype(*i*)**: Type of basis function *i* (*e.g.* X, YZ), meaning of the index is the same as b%type
- **basstart(*i*), basend(*i*)**: Starting and ending basis function indices corresponding to atom *i*

Coefficient matrices of basis functions

- Restricted wavefunction: CObasa(μ, *i*) records coefficient of basis function μ in orbital *i*
- Unrestricted wavefunction: Coefficients of α and β orbitals are recorded in CObasa and CObasb respectively, with the same rule as above

Both CObasa and CObasb have dimension of (nbasis,nbasis)

NOTE: Sometimes there is a linear dependence on the basis functions in quantum chemistry calculation, that is, nindbasis<nbasis, and the number of orbitals of each spin is thus equal to "nindbasis" but not equal to "nbasis". However, in Multiwfn, the number of orbitals for each spin is required to be the same as nbasis. So, at this time, the orbital energies that not loaded are automatically set to 0, and the expansion coefficients that not loaded are automatically set to 0.

Various matrices represented by basis functions

All matrices have dimension of (nbasis,nbasis)

- **Sbas**: Overlap matrix, automatically generated after loading a file containing basis function information, can also be generated by subroutine "genSbas_curr"
- **Ptot/Palpha/Pbeta**: Total, α, β density matrices. For restricted wavefunction only Ptot is defined. Generated by default after loading a file containing basis function information. Can also be generated by subroutine "genP"
- **Tbas**: Kinetic energy integral matrix, can be generated by subroutine "genTbas_curr"
- **FmatA**, **FmatB**: α (or total) Fock matrix, β Fock matrix

- **Velbas**: Velocity vector integral matrix, can be generated by subroutine "genVelbas_curr"
- **Dbas**: Dipole moment integral matrix, can be generated by subroutine "genDbas_curr"
- **Magbas**: Magnetic integral matrix, can be generated by "genMagbas_curr"
- **Quadbas, Octobas, Hexdebas**: Electric quadrupole, octopole and hexadecapole moment integral matrices, can be generated by subroutine "genMultipolebas"

Other noteworthy global variables and arrays

- ifiletype: Type of input file (plain text=0, fch/fchk=1, wfn=2, wfx=3, chg/pqr=4, pdb/xyz=5, NBO .31=6, cub and VASP grid data=7, grd/dx/vti=8, molden=9, gms=10, mol/sdf=11, gjf and ORCA input=12, mol2=13, mwfn=14, gro=15, CP2K input=16, cif=17, POSCAR=18)
- cellv1, cellv2, cellv3: Three translation vectors of cell
- ifPBC: Dimension of present system (0: isolated system, 1/2/3=1/2/3 dimensional system)
- totenergy: Total electronic energy loaded from input file
- virialratio: Virial ratio loaded from input file
- isphergau: If spherical-harmonic basis function is involved
- type2ix, type2iy, type2iz: Converting GTF type index to order of x, y, z of GTF
- GTFtype2name: Converting GTF type index to GTF name

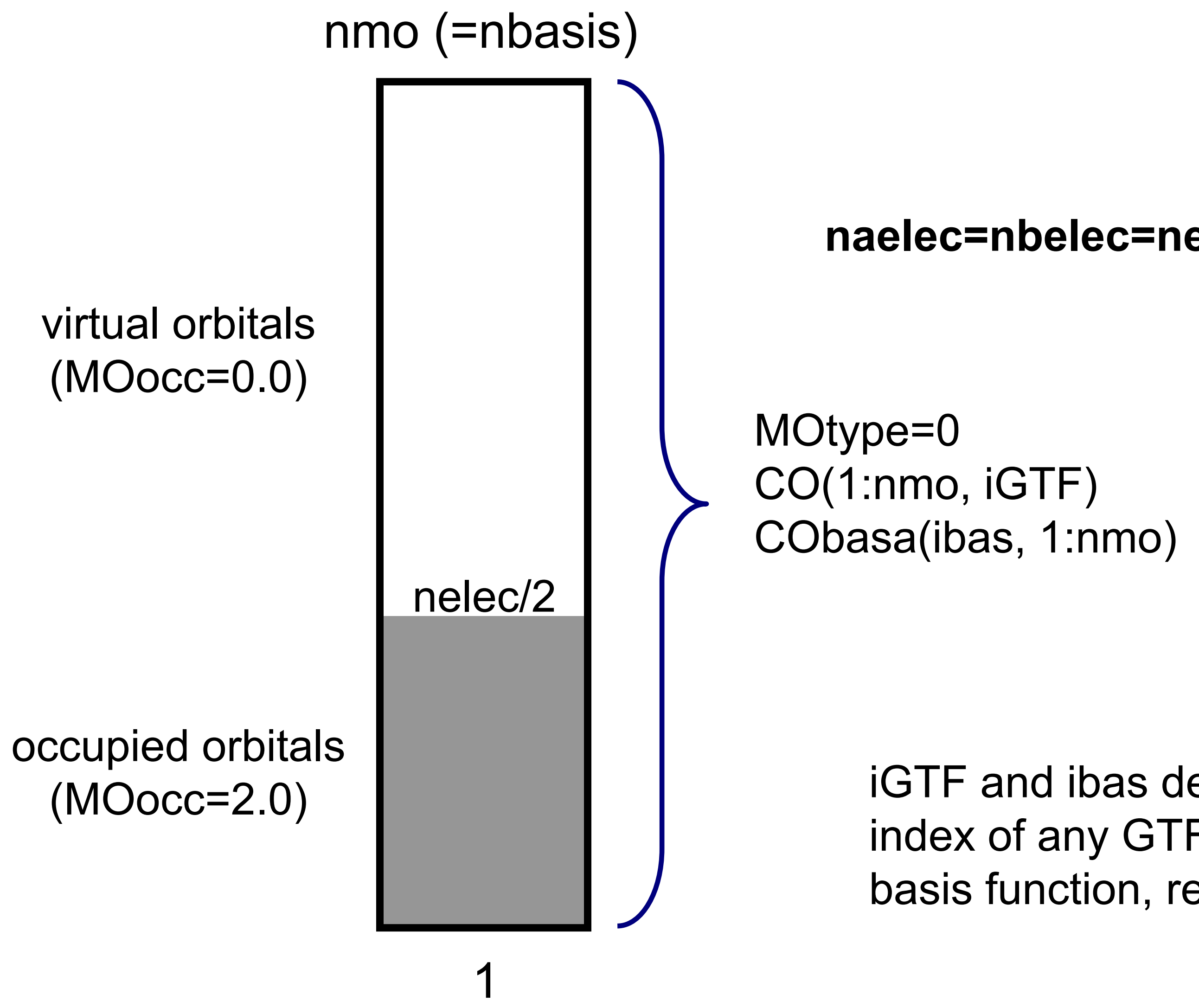
- shtype2name: Converting shell type to shell name
- shtype2nbas: Number of basis functions in each type of shell
- nEDFprims: Number of GTFs of EDF field
- CO_EDF: Orbital expansion coefficient of GTFs of EDF field
- b_EDF: GTF information of EDF field
- filename: File path of currently loaded file
- frag1, frag2: Indices of basis functions in fragments 1 and 2, utilized by a few functions

- nframetraj: Number of frames loaded from .xyz trajectory file
- traj(1/2/3, *a*, *i*): X/Y/Z of atom *a* in frame *i*

The various variables with _org suffix are backups of various information loaded from input file, they are used for restoring original information if they have been changed during analysis. However, the information of the input file loaded first time cannot be restored after loading another input file.

Illustration of recording orbital information

RHF or RKS, wfntype=0

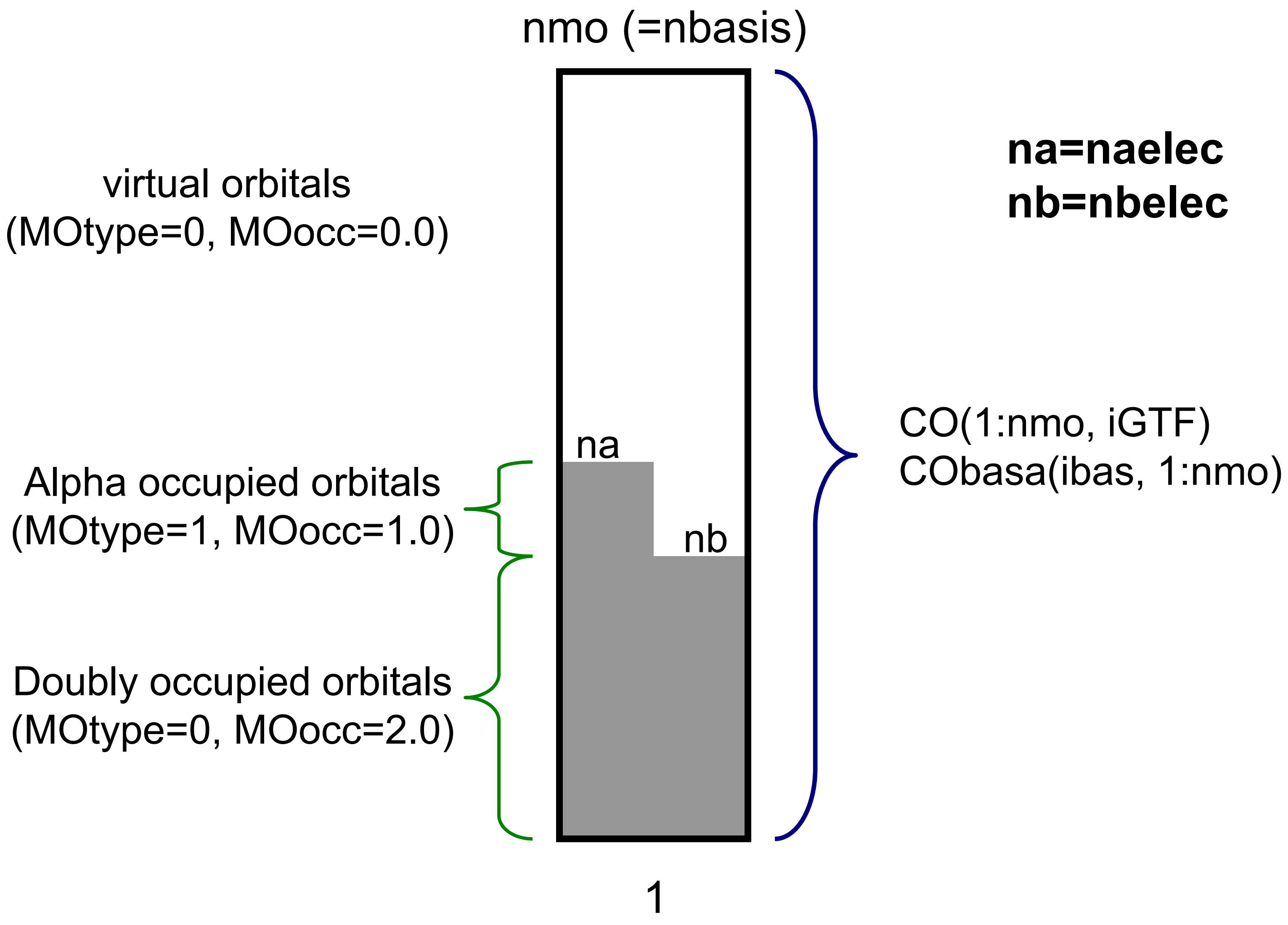


naelec=nbelec=nelec/2

MOtype=0
CO(1:nmo, iGTF)
CObasa(ibas, 1:nmo)

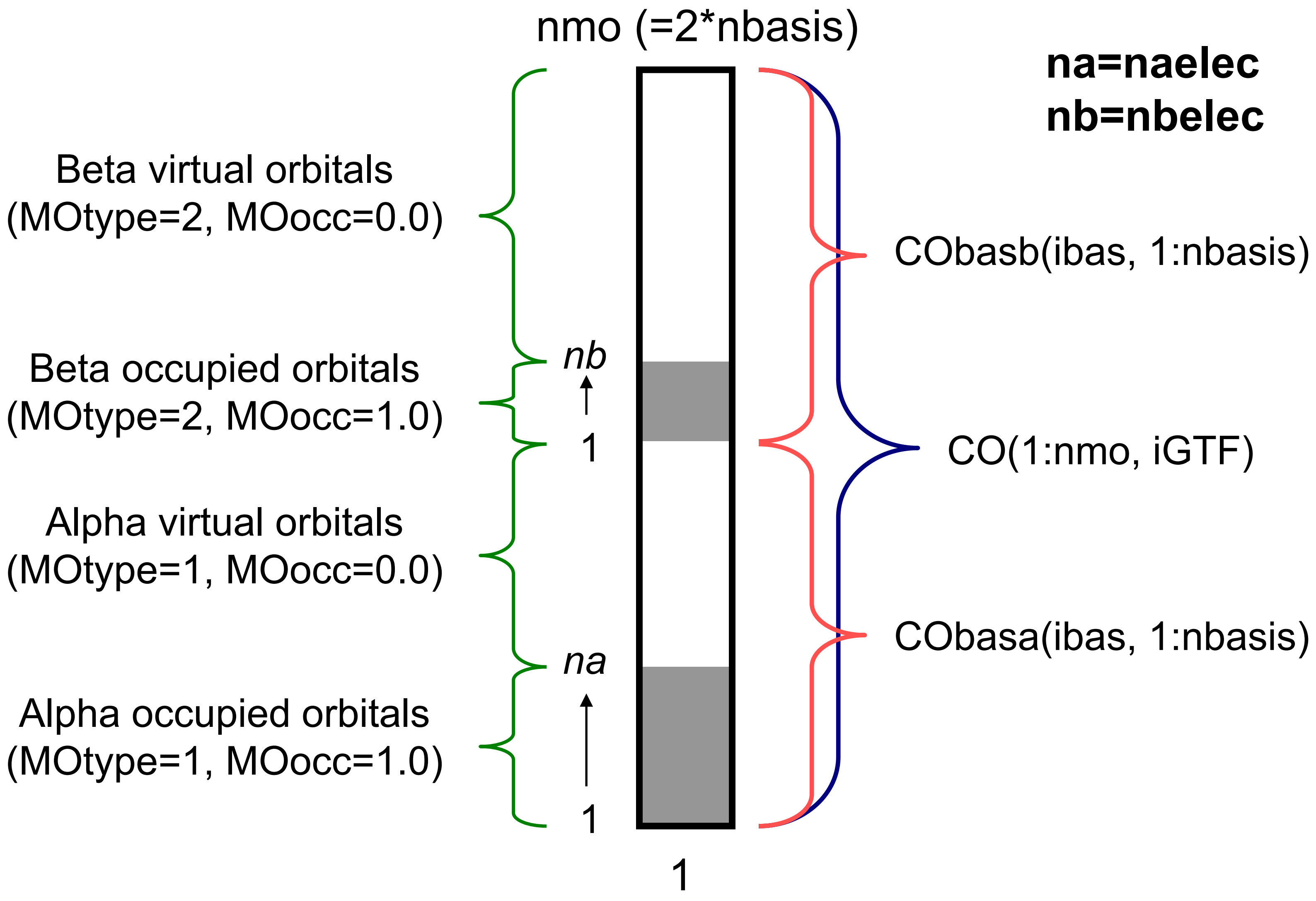
iGTF and ibas denote index of any GTF and basis function, respectively

ROHF or ROKS, wfntype=1

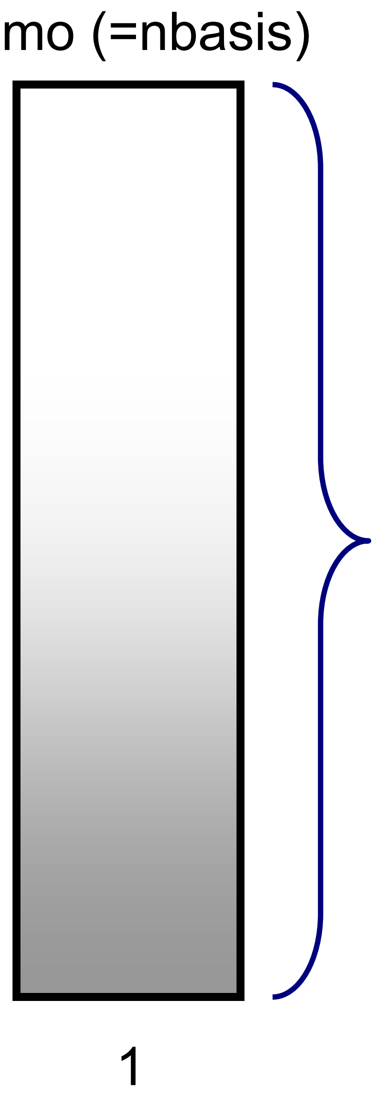


na=naelec
nb=nbelec

UHF or UKS, wfntype=2



Restricted post-HF, wfntype=3

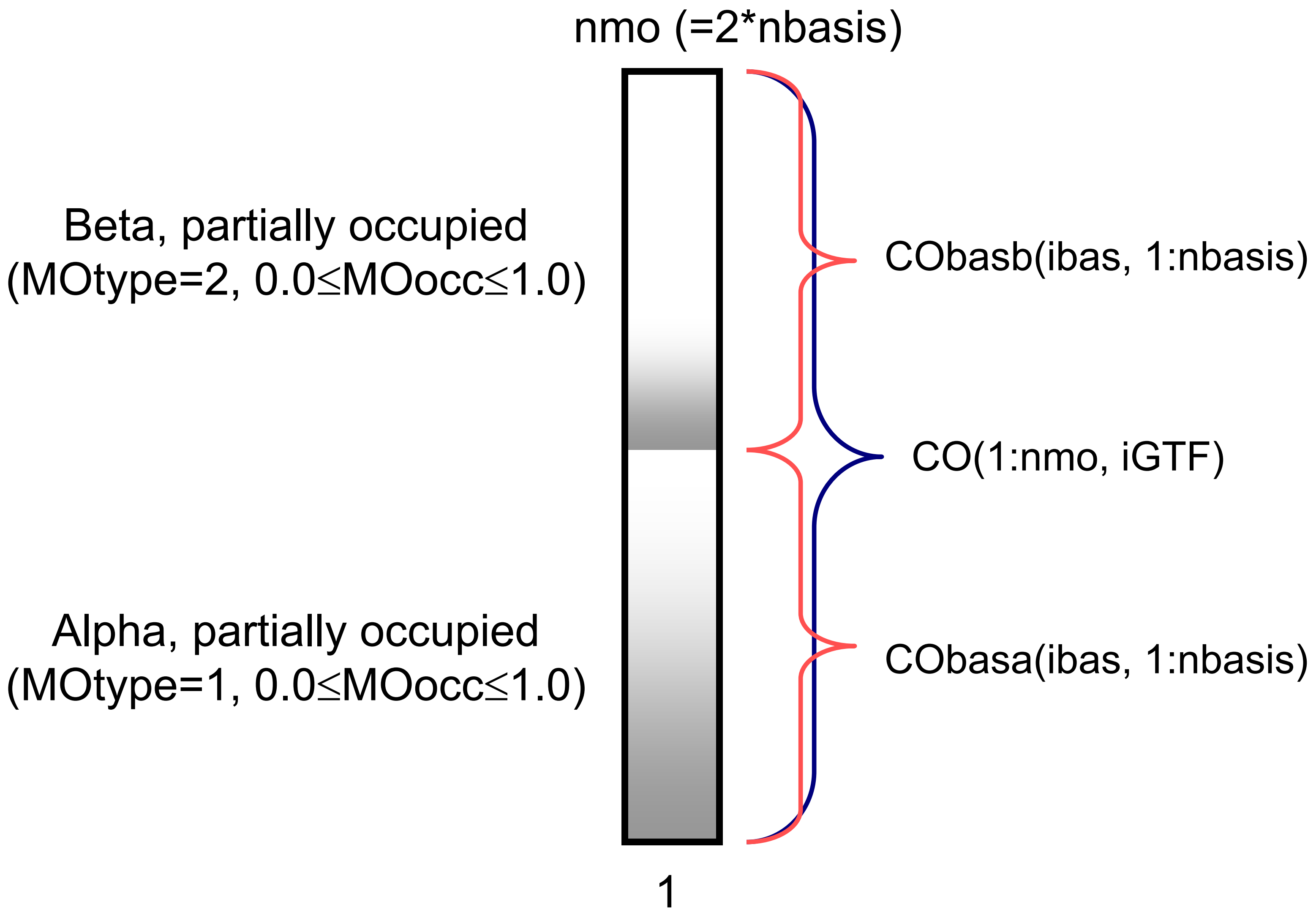


partially occupied
($0 \leq$ MOocc ≤ 2.0)

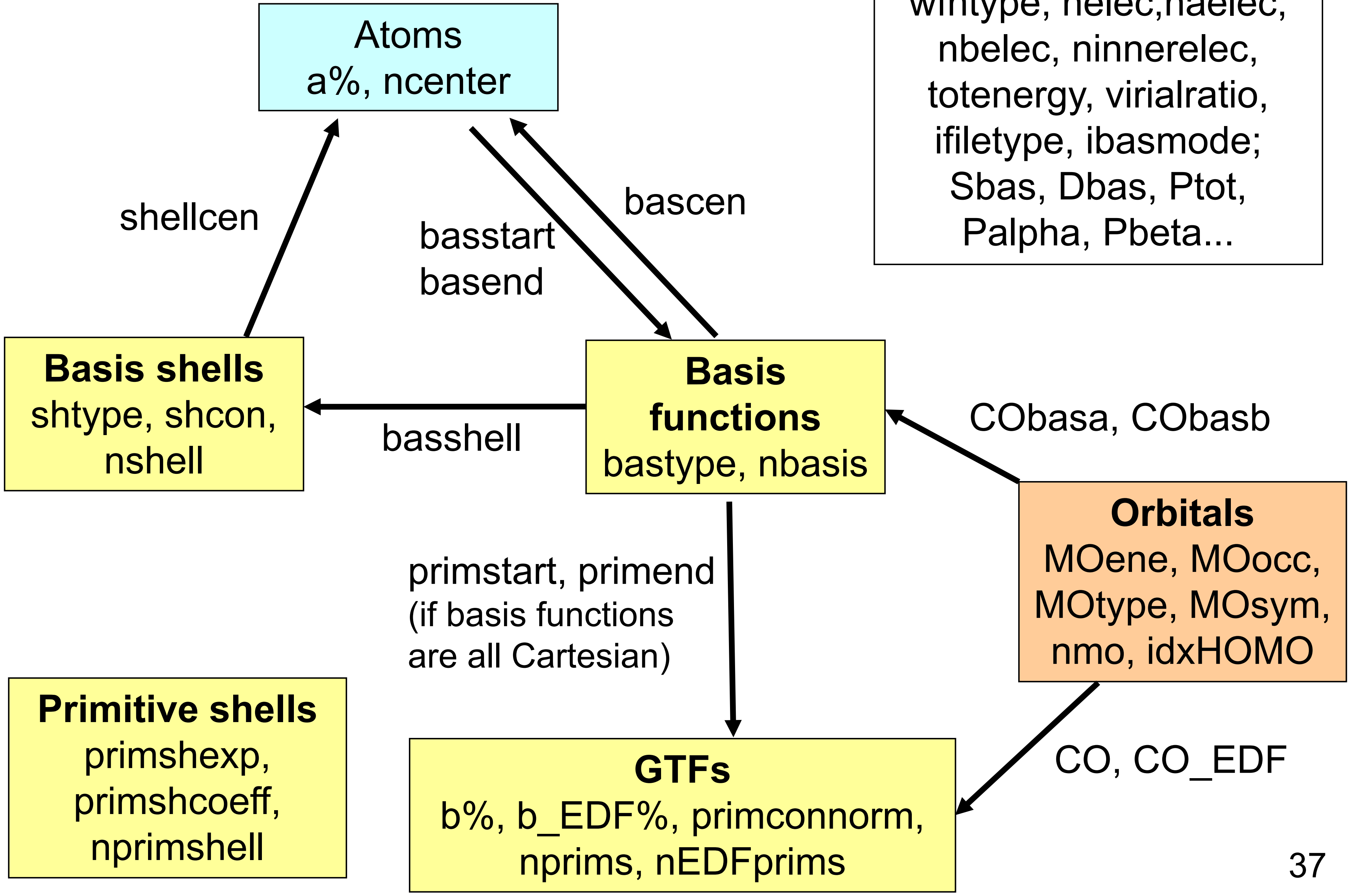
naelec=nbelec=nelec/2

MOtype=0
CO(1:nmo, iGTF)
CObasa(ibas, 1:nmo)

Unrestricted post-HF, wfntype=4



Relationship between variables


Wavefunction
wfntype, nelec,naelec, nbelec, ninnerelec, totenergy, virialratio, ifiletype, ibasmode; Sbas, Dbas, Ptot, Palpha, Pbeta...
Atoms
a%, ncenter
shellcen
basstart
basend
bascen
Basis shells
shtype, shcon, nshell
basshell
Basis functions
bastype, nbasis
CObasa, CObasb
Orbitals
MOene, MOocc, MOtype, MOsym, nmo, idxHOMO
primstart, primend
(if basis functions are all Cartesian)
CO, CO_EDF
Primitive shells
primshexp, primshcoeff, nprimshell
GTFs
b%, b_EDF%, primconnorm, nprims, nEDFprims

Functions and subroutines for evaluating real space functions (in *function.f90*)

More complete list of functions is given in Appendix 2 of Multiwfn manual
Also see comments at beginning of each functions in *function.f90* for more detail

- **function calcfuncall**: A wrapper function for evaluating any real space function at a point
- **function userfunc**: User-defined function, see Section 2.7 of Multiwfn manual for detail
- **function linintp3d**: Calculate function value by trilinear interpolation based on grid data in memory (cubmat)
- **function splineintp3D**: Calculate function value by B-spline interpolation based on grid data in memory (cubmat)
- **function fmo:** Orbital wavefunction value
- **function forbdens:** Orbital probability density
- **function fdens**: Electron density
- **function fspindens**: α or β electron density, or spin density

- **function fgrad**: Gradient of electron density (x/y/z component or norm) or reduced density gradient (RDG)
- **function flapl**: Laplacian of electron density (xx/yy/zz component or total)
- **function Lagkin**: Lagrangian kinetic energy density G(**r**) and its components
- **function Hamkin:** Hamiltonian kinetic energy density K(**r**) and its components
- **function calcprodens**: promolecular density
- **function RDGprodens**: RDG with promolecular approximation
- **function signlambda2rho**: $sign[\lambda_2(\mathbf{r})]*\rho(\mathbf{r})$
- **function signlambda2rho_prodens**: promolecular approximation based $sign[\lambda_2(\mathbf{r})]\rho(\mathbf{r})$
- **subroutine signlambda2rho_RDG_prodens**: Simultaneously calculate RDG and $sign[\lambda_2(\mathbf{r})]*\rho(\mathbf{r})$ under promolecular approximation
- **subroutine IGMprodens**: Normal density gradient or IGM type of density gradient based on promolecular density

- **function ELF_LOL**: ELF, LOL and SCI functions
- **function avglocion**: Averaged local ionization energy
- **function loceleaff**: Local electron affinity
- **function pairfunc**: Exchange-correlation density, correlation hole, correlation factor, on-top pair density
- **function srcfunc**: Source function
- **function delta_g_IGM**: δg function defined by IGM
- **function totesp**: Total electrostatic potential
- **function nucesp**: Electrostatic potential calculated by nuclear charges or atomic charges
- **function eleesp**: Electrostatic potential contributed by electrons
- **function IRIfunc**: IRI function
- **function SEDD**: SEDD function
- **function DORI**: DORI function
- **function energydens_grdn**: Gradient norm of energy density
- **function densellip**: Electron density ellipticity and eta index

- **function elemomdens**: Electron linear momentum density
- **function magmomdens**: Magnetic dipole moment density
- **function edr**: Electron delocalization range EDR(**r**;*d*)
- **function edrdmax**: Orbital overlap distance function *D*(**r**)
- **function PAEM**: Potential acting on one electron in a molecule
- **function infoentro**: Shannon information entropy function or Shannon entropy density
- **function Fisherinfo**: Fisher information density
- **function weizsacker**: Steric energy（=Weizsäcker）
- **function stericpot**: Steric potential
- **function localcorr**: Local electron correlation function
- **function DFTxcfunc**: Integrand of various kinds of DFT exchange-correlation functionals
- **function DFTxcpot**: Various DFT exchange-correlation potentials
- **function KED**: Various kinds of kinetic energy density of electrons

Before using functions or subroutines to evaluate real space functions, "use functions" should be used to load module "functions"。

For functions "totesp" and "eleesp", in *settings.ini* if iESPtot=1, the old and slow code will be used, while in the default case (iESPtot=2), the much faster code based on LIBRETA electron integral library will be used.

When iESPtot=2 and "totesp" or "eleesp" function will be invoked, "use libreta" should be added, and "call initlibreta" must be executed to initialize LIBRETA library before invoking them. Once content of "b" or "CO" array is changed (*e.g.* new file is loaded, wavefunction is modified by main function 6), "call initlibreta" must be executed again.

Example of invoking function:
Evaluating electron density at (1.2,0.0,3.4) Bohr

Add "use functions" to the code, then:

```
tmpval=fdens(1.2,0.0,3.4)
```

or via wrapper function:

```
tmpval=calcfuncall(1,1.2,0.0,3.4)
```

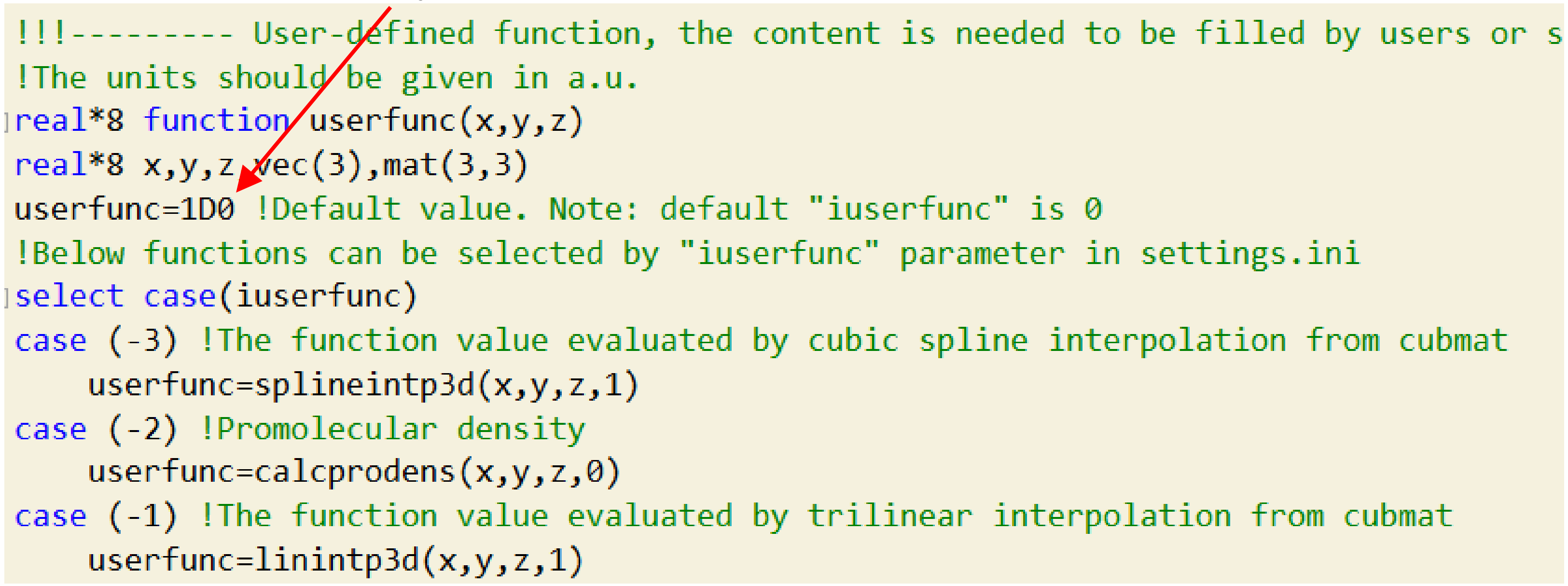
$\uparrow$

Function index of 1 corresponds to electron density, see Section 2.6 of manual for function index

User-defined function

The function "userfunc" in *function.f90* contains many built-in user-defined functions, which can be chosen by "iuserfunc" parameter in *settings.ini*.

By default, the returned value is 1



```
!!!--------- User-defined function, the content is needed to be filled by users or s
!The units should be given in a.u.
real*8 function userfunc(x,y,z)
real*8 x,y,z,vec(3),mat(3,3)
userfunc=1D0 !Default value. Note: default "iuserfunc" is 0
!Below functions can be selected by "iuserfunc" parameter in settings.ini
select case(iuserfunc)
case (-3) !The function value evaluated by cubic spline interpolation from cubmat
    userfunc=splineintp3d(x,y,z,1)
case (-2) !Promolecular density
    userfunc=calcprodens(x,y,z,0)
case (-1) !The function value evaluated by trilinear interpolation from cubmat
    userfunc=linintp3d(x,y,z,1)
```

It is very easy to implement a new real space function in Multiwfn. For example, to realize $xzz\rho(\mathbf{r})$, "userfunc" can be defined as:

userfunc=x*z*z*fdens(x,y,z)

Calculate derivative of real space functions

See comments at the beginning of
corresponding subroutines on how to use

- **subroutine orbderv**: Calculate wavefunction value of a range of orbitals and their derivatives (up to third-order) at a given point
- **subroutine calchessmat_dens**: Calculate electron density, its gradient and Hessian matrix
- **subroutine rho_tensor**: Calculate electron density, its gradient, Hessian matrix and 3rd derivative tensor
- **subroutine calchessmat_prodens**: The same as "calchessmat_dens" but based on promolecular aproximation
- **subroutine EDFrho**: Calculate contribution from EDFs to density and corresponding derivatives (up to third-order)
- **subroutine gendensgradab**: Generate electron density and gradient norm for alpha and beta electrons at the same time
- **subroutine gendens_gradvec_lapl_ab**: Generate electron density, gradient vector and Laplacian for alpha and beta electrons at the same time

- **subroutine proatmgrad**: Calculate electron density and gradient of an atom in free state using built-in density library
- **subroutine calchessmat_lapl**: Calculate value and gradient of Laplacian of electron density
- **subroutine calchessmat_ELF_LOL**: Calculate value and gradient of ELF and LOL
- **subroutine calchessmat_orb**: Calculate gradient and Hessian matrix for an orbital wavefunction
- **subroutine calchessmat_rhograd**: Calculate gradient and Hessian matrix for gradient norm of electron density
- **subroutine calchessmat_IRI_RDG**: Calculate gradient and Hessian matrix for IRI and RDG
- **subroutine gencalchessmat**: A general subroutine to calculate value, gradient and Hessian for various real space functions

Some other real space functions also have corresponding subroutines to calculate analytic derivatives, see their invokings in gencalchessmat

Derivative of real space functions in Multiwfn

- Analytic gradient and Hessian:
  orbital wavefunction, electron density (including promolecular version), IRI, RDG, vdW potential, local information entropy & Shannon entropy density, relative shannon entropy density, Fisher information density
- Analytic gradient + Numerical Hessian:
  Laplacian of electron density, ELF, LOL, energy density, second Fisher information density, steric potential
- Numeric gradient and Hessian: Most other real space functions

Example:
Calculate value, gradient and Hessian of electron density at (1.2, 0.0, 3.4) Bohr

Use subroutine dedicated to electron density evaluation:

```
call calchessmat_dens(2, 1.2, 0.0, 3.4, value, grad, hess)
```

or use general subroutine:

```
call gencalchessmat(2, 1, 1.2, 0.0, 3.4, value, grad, hess)
```

1: Calculate value and gradient
2: Calculate value, gradient and Hessian

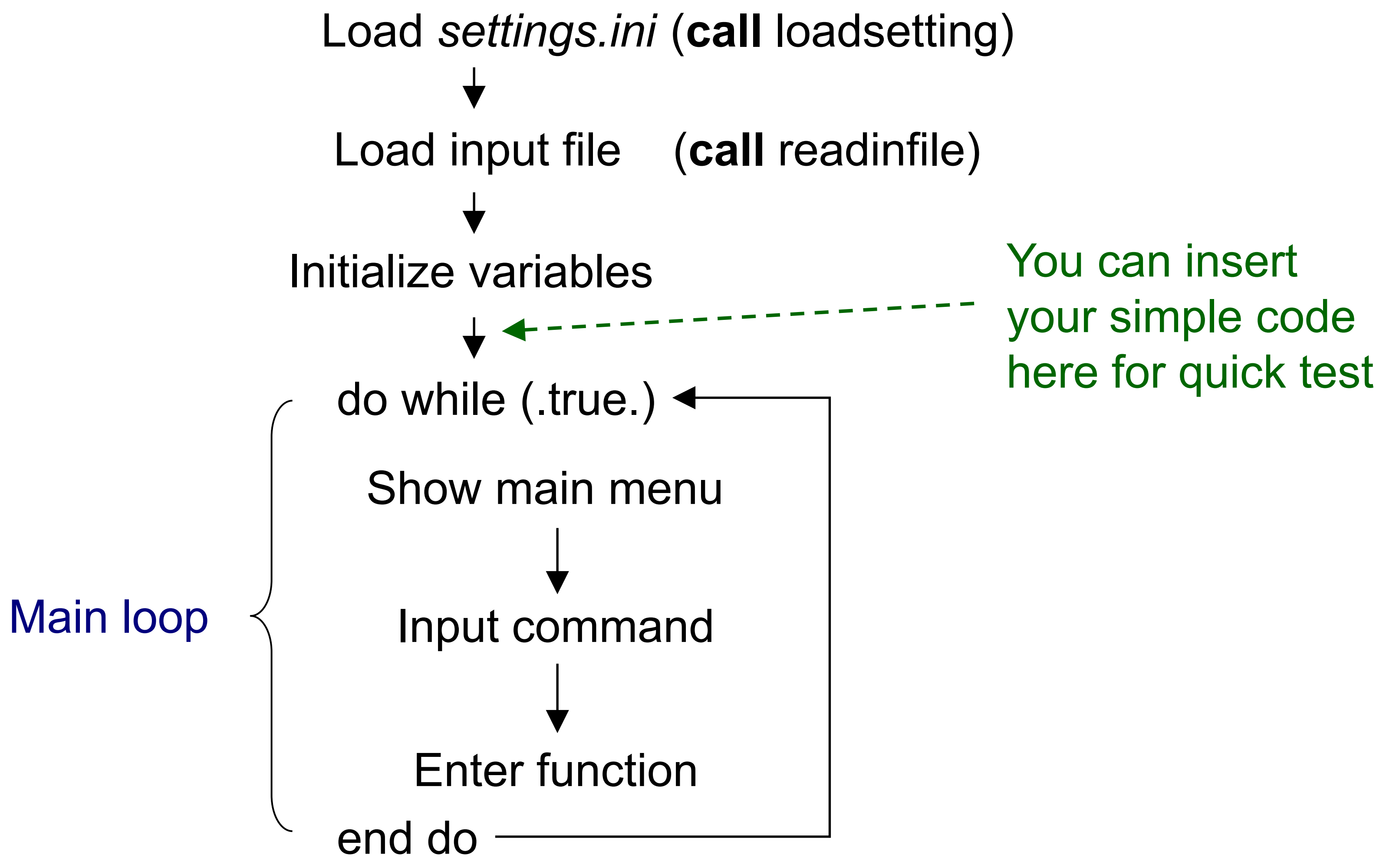
1: Electron density

value: scalar
grad: vector(3)
hess: matrix(3,3)

2

Example of adding new functions

The running process of Multiwfn (*Multiwfn.f90*)


Load *settings.ini* (**call** loadsetting)
Load input file (**call** readinfile)
Initialize variables
You can insert your simple code here for quick test
do while (.true.)
Show main menu
Input command
Enter function
end do
Main loop

Example 1:
Output spin density divided by total density at every nuclear position

```
subroutine exam_1
use defvar
use functions
implicit real*8 (a-h,o-z)
do iatm=1, ncenter
  value1=calcfuncall(5,a(iatm)%x,a(iatm)%y,a(iatm)%z)
  value2=calcfuncall(1,a(iatm)%x,a(iatm)%y,a(iatm)%z)
  write(*,"(i6,'(',a,'):', f12.6)") iatm,a(iatm)%name,value1/value2
end do
end subroutine
```

"exam_1" can be called at proper place,
e.g. before entering main loop in *Multiwfn.f90*.

Result

1(C) :	0.003154
2(H) :	-0.010976
3(N) :	0.000230
4(H) :	0.052122
5(H) :	0.010877
6(O) :	0.001042

Example 2:
Output orbital composition for all singly occupied orbitals via SCPA method

```
subroutine exam_2
use defvar
implicit real*8 (a-h,o-z)
do imo=1,nmo
  if (MOtype(imo)==1) then
      write(*,"(/,' Composition of MO',i6,' Occ:',f10.6)") imo,MOocc(imo)
      sumsqrall=sum(CObasa(:,imo)**2)
      do iatm=1,ncenter
          istart=basstart(iatm)
          iend=basend(iatm)
          value=sum(CObasa(istart:iend,imo)**2)/sumsqrall*100
          write(*,"(i6,'(',a,'):',f12.6,'%')") iatm,a(iatm)%name,value
      end do
  end if
end do
end subroutine
```

Result (H_2O, triplet, ROHF/STO-3G)

```
Composition of MO     5 Occ:  1.000000
    1(O ) :  100.000000%
    2(H ) :    0.000000%
    3(H ) :    0.000000%

Composition of MO     6 Occ:  1.000000
    1(O ) :   49.478230%
    2(H ) :   25.260885%
    3(H ) :   25.260885%
```

Example 3: Using Mulliken method to calculate total number of p_z electrons on all carbon atoms

```
subroutine carbonZpop
use defvar
implicit real*8 (a-h,o-z)
real*8 PSmat(nbasis,nbasis)
Zpop=0
PSmat=Ptot*Sbas
do iatm=1,ncenter
    if (a(iatm)%index/=6) cycle
    do ibas=basstart(iatm),basend(iatm)
        if (bastype(ibas)==4) Zpop=Zpop+sum(PSmat(ibas,:))
    end do
end do
write(*,"(' Number of pz electrons on carbons:',f10.6)") Zpop
end subroutine
```

Example 4:
Add harmonic function $0.005*|r-r_0|^2$ to loaded grid data

(where origin r_0 is geometry center of present system)

```
subroutine exam_4
use defvar
implicit real*8 (a-h,o-z)
cenx=sum(a(:)%x)/ncenter
ceny=sum(a(:)%y)/ncenter
cenz=sum(a(:)%z)/ncenter
do iz=1,nz
    do iy=1,ny
        do ix=1,nx
            call getgridxyz(ix,iy,iz,x,y,z)
            value=0.005D0*((x-cenx)**2+(y-ceny)**2+(z-cenz)**2)
            cubmat(ix,iy,iz)=cubmat(ix,iy,iz)+value
        end do
    end do
end do
end subroutine
```

Then, export updated grid data to a cube file

```
open(10,file="C:\abc.cub",status="replace")
call outcube(cubmat,nx,ny,nz,orgx,orgy,orgz,
dx,dy,dz,gridvec1,gridvec2,gridvec3,10)
close(10)
```

Alternatively, export grid data by user manually via option 0 in main function 13