

Representation and conversion of the set of wavefunctions

(X. Gonze, Y. Suzukawa, M. Mikami)

April 10, 2004

1 The block of wavefunctions for one **k**-point and one spin-polarization

* We will now consider a wavefunction as an object made of $2*\text{npw_k}*\text{nspinor}$ double precision numbers, whose signification and use will not be described here.

* npw_k is an integer number, that may vary from **k**-point to **k**-point, while nspinor will be 1 or 2, and will not vary in the set of wavefunction.

* A block of wavefunction is made of nband_k wavefunctions, considered for one specific **k**-point and spin-polarization. The number of double precision coefficients in a block of WFs is $2*\text{npw_k}*\text{nspinor}*\text{nband_k}$.

* The set of wavefunctions is made of all the blocks for different **k**-points and spin-polarizations. The number of spin-polarization, nsppol , can be 1 or 2. Note that $\text{nsppol}=2$ and $\text{nspinor} = 2$ are mutually exclusive. The number of **k**-points, nkpt can be as small as 1, but can also be a large number, bigger than 1000. There must be the same number of **k**-points for both spin-polarizations.

* As npw_k and nband_k can vary with the **k**-point (and polarization for nband_k), we have arrays

$$\begin{aligned}\text{npwarr}(1 : \text{nkpt}, 1) &\longrightarrow \text{npw_k} = \text{npwarr}(\text{ikpt}, 1) \\ \text{nband}(1 : \text{nkpt} * \text{nsppol}) &\longrightarrow \text{nband_k} = \text{nband}(\text{ikpt} + (\text{isppol} - 1) * \text{nkpt})\end{aligned}$$

2 Eigenvalues and occupation numbers

* At each **k**-point and spin-polarization, there is also a set of eigenvalues and a set of occupation numbers, in the Ground-State case (`formeig = 0`) ;

```
eig_k(1:nband_k)
```

```
occ_k(1:nband_k)
```

and, in the Response-Function case (`formeig = 1`), a complex matrix of eigenvalues

```
eig_k(1:2*nband_k**2)
```

3 Storage of wavefunctions : disk file

The disk files are made of a header, followed by the blocks of wavefunctions, eigenvalues (and occupation numbers, in the ground-state case) for each **k**-point and spin-polarization, then some information on the history of atomic positions and forces.

The part related to the wavefunctions block is written as follows :

```
do isppol= 1, nsppol
```

```
do ikpt = 1, nkpt
```

```
write(unit) npw_k*nspinor, nband_k
```

```
if(formeig == 0) then
```

```
write(unit) eig_k(1:nband_k), occ_k(1:nband_k)
```

```
end if
```

```
do iband = 1, nband_k
```

```
if(formeig == 1) then
```

```
write(unit) eig_k(1:2*nband_k*nband_k)
```

```
end if
```

```
write(unit) wavef_k(1:2, 1:npw_k*nspinor, iband)
```

```
enddo ! iband
```

```
enddo ! ikpt
```

```
enddo ! isppol
```

where :

`formeig = 0` for ground-state wfs, and `= 1` for response function

`npw_k`, `nband_k`, `eig_k`, `occ_k`, `wavef_k` are related to one **k**-point and spin-polarization, and vary with them (this is not shown explicitly in the above description).

4 Storage of wavefunctions : core memory (sequential case)

* In order to describe the storage of wavefunctions, we adopt the same convention on the meaning of `npw_k`, `nband_k`, `eig_k`, `occ_k` and `wavef_k`.

We have to distinguish two cases : either the full set of wave function is kept in memory ($m_{\text{mem}} = n_{\text{kpt}}$), or only one block of wavefunction is kept in memory ($m_{\text{mem}} = 0$). The intermediate case, were a subset of the wavefunctions would be kept in core memory, has no advantage with respect to one or the other, and has not been allowed.

* If $m_{\text{mem}} = n_{\text{kpt}}$, the wavefunctions are kept in the array `cg`, declared as

```
double precision :: cg(2,mpw*nspinor*mband*mkmem*nsppol)
```

where `mpw` is the maximum number of plane waves for all **k** points
`mband` is the maximum number of bands for all **k** points.

The detailed storage is :

```
icg = 0
do isppol = 1, nsppol
do ikpt = 1, nkpt
do iband = 1, nband_k
do ipwsp = 1, npw_k*nspinor
index = ipwsp + (iband - 1) * npw_k * nspinor
cg(1:2, index + icg) = wavef_k(1:2, ipwsp, iband)
enddo
enddo
icg = icg + npw_k * nspinor * nband_k
enddo
enddo
```

* If $m_{\text{mem}} = 0$, the wavefunctions are kept on disk, and the block of wavefunctions related to one **k**-point and spin-polarization is read in the array `cg_disk`, declared as

```
double precision :: cg_disk (2, npw_k * nspinor * nband_k)
```

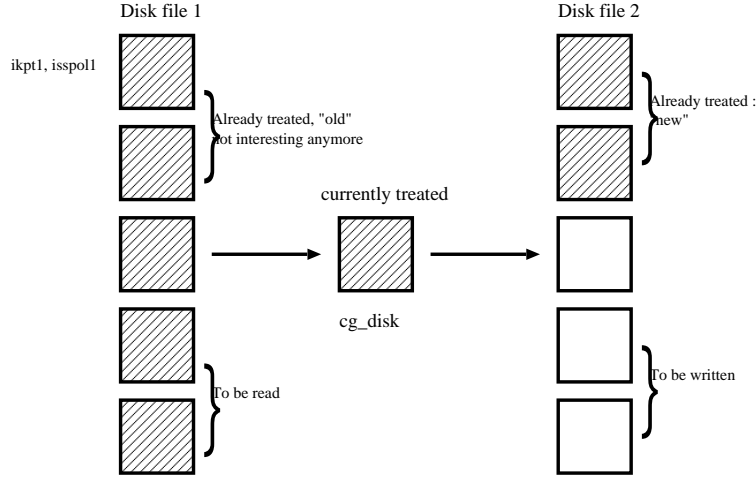
reallocated for each block, with the correct dimensions.

The self-consistent update of wavefunctions will actually involve two disk files. During one “step” of the SCF procedure, the “old” wavefunctions are contained in a first disk file, that is read block of wavefunctions by block of wavefunctions, while the “new” wavefunctions are written on another disk file :

* Independently of the value of m_{mem} , the eigenvalue and occupation numbers are kept in core memory.

For the occupation numbers, one has the array `occ`

```
double precision :: occ (mband * nkpt * nsppol)
with detailed storage
bantot = 0
do isppol = 1, nsppol
do ikpt = 1, nkpt
occ (1 + bantot : nband_k + bantot) = occ_k (1:nband_k)
bantot = bantot + nband_k
enddo
enddo
```



* The storage of eigenvalues, in the ground-state case (`formeig = 0`) is perfectly identical to the one of occupation numbers, in the array `eigen` :

```
double precision :: eigen (mband * nkpt * nsppol)
```

For the response-function case, we have matrices of eigenvalues :

```
double precision :: eigen (2 * mband **2 * nkpt * nsppol)
```

```
ban2tot = 0
```

```
do isppol = 1, nsppol
```

```
do ikpt = 1, nkpt
```

```
eigen(1+ban2tot: 2*nband_k**2 + ban2tot) = eig_k(1:2*nband_k**2)
```

```
ban2tot = ban2tot + 2*nband_k**2
```

```
enddo
```

```
enddo
```

5 Storage of wavefunctions : core memory (parallel case)

* In the parallel case, the storage of wavefunctions is spread on the different processors, while all processors have a copy of the arrays `eigen` and `occ`, whose storage is not modified compared to the sequential case. We will thus focus on the wavefunctions.

* The fundamental question is : does the present processor treat this **k**-point and spin-polarization ? If yes, the corresponding block will be in core memory. If no, it will not. In the parallel case, it might be interesting to have `mkmem` lower than `nkpt`, as soon as the number of **k** points to be treated by a processor

```

is lower or equal to mkmem. We still have the array cg declared as
double precision :: cg(2, mpw * nspinor * mband * mkmem * nsppol)
with detailed storage :
icg = 0
do isppol = 1, nsppol
do ikpt = 1, nkpt
if ((' (ikpt, isppol) not treated by the present processor')) cycle
do iband = 1, nband_k
do ipwsp = 1, npw_k * nspinor
index = ipwsp + (iband-1) * npw_k * nspinor
cg (1:2,index + icg) = wavef_k (1:2, ipwsp, iband)
enddo
enddo
icg = icg + npw_k * nspinor * nband_k
enddo
enddo

```

* Let us specify the meaning of “(ikpt, isppol) not treated by the present processor”

There are two parallel modes. Either the parallelism is done with respect to **k**-points only, or it is allowed with respect to **k**-points and bands. In the present status of ABINIT (v3.2), in the ground-state case, the parallelism is done only with respect to **k**-points, while in the response-function case, it is done with respect to **k**-points and bands. The user has no control yet on this choice.

* The case `paralbd = 0` (no parallelism over the bands)
(Warning : Should be updated ! similar to the case `paralbd = 1` in v3.2).
The attribution of a **k**-point for some spin-polarization is computed in the routine `distrb.f`, and generates an array `kpt_distrib(1:nkpt)` giving for each **k**-point, the number of the processor that treats it. This number, for each processor, is obtained through a call to the MPI routine `MPI_COMM_RANK`, and stored in the variable `me`. The condition for this **k**-point `ikpt` to be treated by the processor `me` is thus :
`if (kpt_distrib(ikpt) == me) then ...`

* The case `paralbd = 1` (parallelism over bands is allowed)
In this case, some bands of a same **k**-point can be treated by different processors. However, for different reasons, the processors that treat bands belonging to a **k**-point need to know all the wavefunctions of this **k**-point. Thus one block of wavefunction for one **k**-point will be copied on different processors.

The attribution of a band of some **k**-point, for some spin-polarization, is computed in the routine `distrb2.f`, and generates an array `proc_distrb (1:nkpt, 1:mband)` for each spin-polarization, giving for each **k**-point and band, the number of the processor that treats it.

The condition, for the processor `me`, to contain the block of wavefunctions associated to the **k**-point `ikpt`, is to have at least one band attributed to it :

```
if (minval(abs(proc_distrb(ikpt, 1:nband_k)-me))=0) then ...
(this is a very condensed F90 formulation !)
```

6 Reading and Conversion of wavefunctions : principles [routine inwffil.f]

* We have seen in the document Data structures 1WF, page 7, how to derive the wavefunction characterized by `nspinor`, `kpt`, `kg`, and `istwfk`, from some other wavefunction with different parameters. Here, we consider the conversion of blocks of wavefunctions, for which the additional parameters `nband_k`, `nkpt` and `nsppol` are present, and can be varied.

* Typically, the starting wavefunctions are on a disk file, and they must generate other wavefunctions, either in core memory or on another disk file. The treatment will differ in those two cases, especially because of the core memory management. In this operation, the goal is not to use more memory than in the rest of the code! Efficiency is a secondary concern, in the sequential version. It is more important in the parallel version, however, as we will see later.

* Thus, it is not possible to create two arrays with dimensions
`cg1 (2, mpw1 * nspinor1 * mband1 * mkmem1 * nsppol1)`
`cg2 (2, mpw2 * nspinor2 * mband2 * mkmem2 * nsppol2)`
and make the conversion in core memory :

$$\begin{array}{ccccc} \text{disk} & \xrightarrow{\text{rendering}} & \text{cg1} & \xrightarrow{\text{converting}} & \text{cg2} & \leftarrow \text{This is not possible !} \\ & & & & \downarrow & \text{possibly write} \\ & & & & \text{disk} & \end{array}$$

* The precise mechanism, with temporary arrays, will depend on the parameters `mkmem`, the sequential/parallel mode as well as the relative sizes of the “input” and “output” blocks of wavefunctions for each **k**-point and spin-polarization. See section 7 and 8.

* Independently of this mechanism, we describe the change of parameters `kpt`, `nband_k` and `nsppol` now. Be given an input set of `nkpt1` **k**-point wavefunctions, with **k**-points `kpt1(1:3, 1:nkpt1)`, from which the wavefunctions at `nkpt2` **k**-points `kpt2(1:3, 1:nkpt2)` must be deduced. For each **k**-point `ikpt2`, we will find the **k**-point `ikpt1` that allows to generate the closest **k**-point, by use of symmetry operations and translations in reciprocal space, as explained in Data structures 1WF page 8. This operation is done in the routine `listkk.f`.

* Having assigned one `ikpt1` to each `ikpt2`, we will also have to select the proper spin-polarization `isppol`. When `nsppol1 = nsppol2`, there is no problem, as `isppol1 = 1` goes to `isppol2 = 1`, and, if `nsppol1 = 2`, we also have `isppol1 = 2` goes to `isppol2 = 2`.

If `nsppol1 = 1` and `nsppol2 = 2`, we will simply use `isppol1 = 1` for both `isppol2 = 1` and `isppol2 = 2`. [the conversion from spinor `wf` to spin-polarization `wfs` is not coded yet]

If `nsppol1 = 2` and `nsppol2 = 1` (reduction from spin-polarized to spin-unpolarized), we use `isppol1 = 1` for `isppol2 = 1`. [the conversion from spin-polarized `wfs` to spinor `wfs` is not coded yet]

* The number of bands needs to be treated as well. From `nband1_k` starting bands, one can generate at most `nband12 = (nband1_k/nspinor1) * nspinor2` output bands, using the mechanism explained in Data structure 1WF, page 7, if conversion of `nspinor` is needed.

We can have three cases : either

- `nband12 = nband2_k` In this case, we use all the input wavefunctions, and generate all the wavefunctions that are needed.
- `nband12 > nband2_k` We have too many input bands, for the restricted number that we need. We actually will read only `nband12_min = (nband2_k/nspinor2) * nspinor` bands.
- `nband12 < nband2_k` We do not have enough starting bands. We will complete the available data either by random numbers (if GS case) or zeros (if RF case)

* The conversion of a block of wavefunctions to another block of wavefunctions is done in the routine `wfconv.f`.

7 The reading and conversion of wavefunction sets. Sequential case.

* We have to distinguish two cases : either the final wavefunctions must be stored on disk, or they will be in core memory.

* Final storage on disk (from disk to disk) [routine `newkpt.f`] A temporary array, dimensioned so as to be able to contain the biggest block of wavefunctions of both the input and output files is created :

```
cg_disk (2, mpw * mspinor * mband)
```

Note that `mband` does not take into account `nband1_k`, when it is sufficient to use `nband12_min`, see page 7.

The blocks (ikpt2, isppol2) will be treated in order, while the blocks (ikpt1, isppol1) will be accessed “randomly”, as needed to obtain all the (ikpt2, isppol2) in turn: do isppol2 = 1, nsppol2
do ikpt2 = 1, nkpt2
· select (ikpt1, isppol1) needed for (ikpt2, isppol2)
· read the wavefunction (ikpt1, isppol1) from disks and store them in cg_disk
· convert the wavefunctions inside cg_disk to (ikpt2, isppol2)
· write the wavefunctions to disk2
enddo
enddo

* Final storage in core memory (from disk to core, routines wfsinp.f and newkpt.f)

In this case, we have an array

cg2(2, mpw2 * nsinor2 * mband2 * mkmem2 * nsppol2)

We will be able to avoid declaring a temporary array cg_disk only if each of the block of input wavefunctions needs less storage than each corresponding block in cg2, that is $2 * npw2_k * nspinor2 * band2_k$ double precision numbers. In this case squeeze = 0, otherwise squeeze = 1. Unlike the disk to disk case, the order in which the input wavefunctions are read is not dictated by the need to write the output wavefunctions in order. On the contrary, we will be able to read each input wavefunction block only once. The algorithm is as follows :

```

Step1 (routine wfsinp.f)
do isppol1 = 1, nsppol1
do ikpt1 = 1, nkpt1
if ‘(ikpt1, isppol1) is needed to initialize some (ikpt2, isppol2)’
then
· read (ikpt1, isppol1), store it in
cg_disk if squeeze =1
cg2(ikp2_stor, isppol2_stor) if squeeze =0
do isppol2=1, nsppol2
do ikpt2=1, nkpt2
if squeeze = 0, copy from (ikpt2_stor, isppol2_stor) to (ikpt2, isppol2)
if squeeze = 1, copy from cg_disk to (ikpt2, isppol2)
enddo
enddo
else skip (ikpt1, isppol1)
endif
enddo
enddo

```

```

Step2 (routine newkpt.f) do isppol2 = 1, nsppol2
do ikpt2 = 1, nkpt2
if squeeze = 0, convert the wavefunctions in block (ikpt2, isppol2)

```



```

to their final parameters
if squeeze = 1, do nothing (the conversion already took place in wfsinp.f)
enddo
enddo

```

8 The reading and conversion of wavefunction sets. Parallel case

* In addition to the disk vs core memory choice, we have to distinguish the case of “local input wavefunction files” vs “unique input wavefunction files”. The input variable `localrdwf` can be used to select one mode or the other.

* In the first mode, `localrdwf = 1` (local input wavefunction files), it is supposed that each processor will access directly the input wavefunction file, either because all processors have access to the same disk system on which one copy of the file exist, or because a copy of the file has been placed on the disk system of each processor. For a SMP machine, the access to the file by different processors could compete and cause a degradation of performance. For a cluster, there is no such problem, but a copy of the input file must be placed on the local disk of each processor beforehand.

The organization of the reading and conversion is very similar to the sequential case. The major difference lies in the question : “does the present processor treat this k -point and spin-polarization?” as explained in pages 4 to 5. The appropriate selection rules, based on `kpt_distrib` or `proc_distrib`, will be used.

* In the second mode, `localrdwf = 0` (“unique input wavefunction file”), it is supposed that only one processor will read the input wavefunction file, and will transmit the information to the other processors. This mode can be more efficient on an SMP machine, and needs less file manipulation for a cluster. However it has only been coded for the “disk to core memory” case.

Coming back to the 2 steps explained in pages 8 and 8, the first one will be strongly modified, while the second step will be adopted to whether the k -point and spin-polarization belongs to the processor.

The temporary array `cg_disk` will always be defined, and used for the transfers of wavefunction blocks.

```

Step 1 (routine wfsinp.f)
For processor me = 0
do isppol1 = 1, nsppol1
do ikpt1 = 1, nkpt1
if ‘(ikpt1, isppol1) is needed to initialize some (ikpt2, isppol2)’
then
· read (ikpt1, isppol1), store it in cg_disk
· send it to all processors that need (ikpt1, isppol1) to

```

```

initialize one of their (ikpt2, isppol2)
do isppol2 = 1, nsppol2
do ikpt2 = 1, nkpt2
if '(ikpt2, isppol2) belongs to me and come from (ikpt1,isppol1)''
then
if (squeeze = 0), copy from cg_disk to (ikpt2, isppol2)
if (squeeze = 1), convert from cg_disk to (ikpt2, isppol2)
end if
enddo
enddo
end if
enddo
enddo
      (step 1)
For processor me  $\neq$  0
do isppol = 1, nsppol1
do ikpt1 = 1, nkpt1
if '(ikpt2, isppol1) is needed to initialize some of my (ikpt2, isppol2)''
then
· receive (ikpt1, isppol1) from processor me = 0
do isppol2 = 1, nsppol2
do ikpt2 = 1, nkpt2
if '(ikpt2, isppol2) belongs to me and come from (ikpt1, isppol1)''
then
if (squeeze = 0), copy from cg_disk to (ikpt2, isppol2)
if (squeeze = 1), convert from cg_disk to (ikpt2, isppol2)
end if
enddo
enddo
end if
enddo
enddo
      Step 2 (routine newkpt.f) for all processors
do isppol2 = 1, nsppol2
do ikpt2 = 1, nkpt2
if '(ikpt2, nsppol2) is treated by me'' then
if squeeze = 0, convert the wavefunction in block (ikpt2, isppol2)
to their final parameters
if squeeze = 1, do nothing (the conversion already took place in wfsinp.f)
end if
enddo
enddo

```