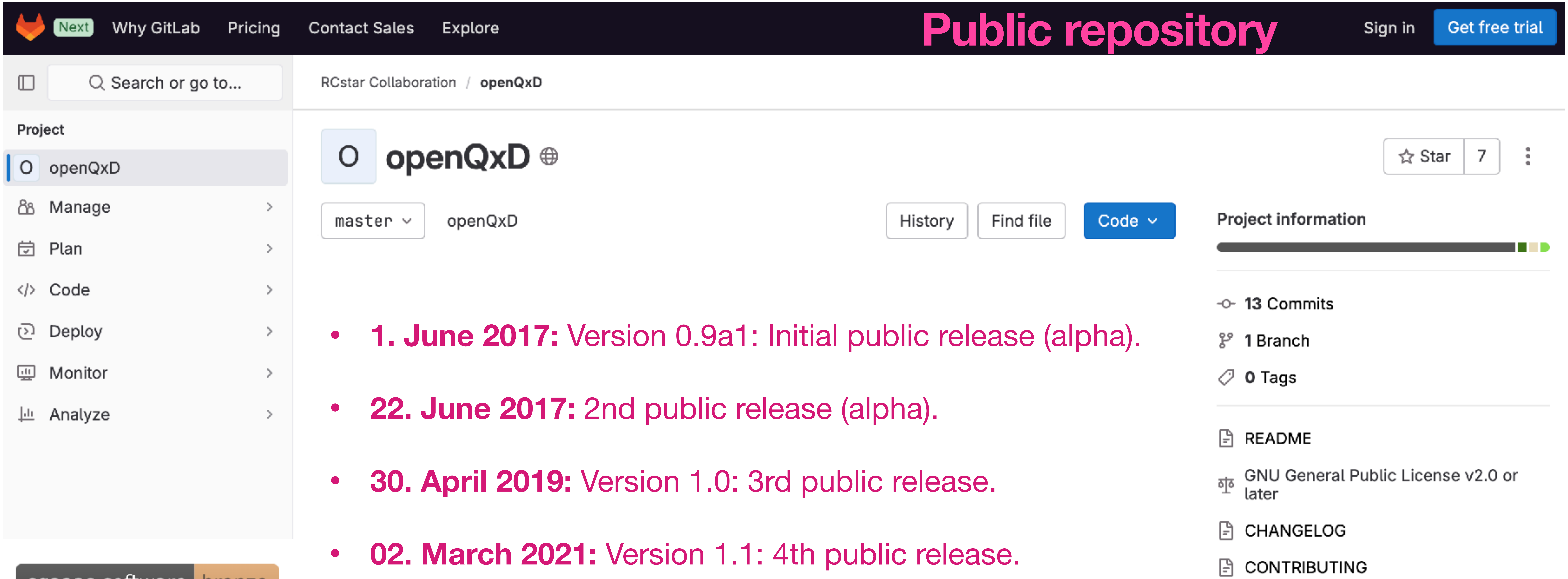
openQ*D code

Thoughts on future releases and development

Agostino Patella, 20.01.2024

Public repository



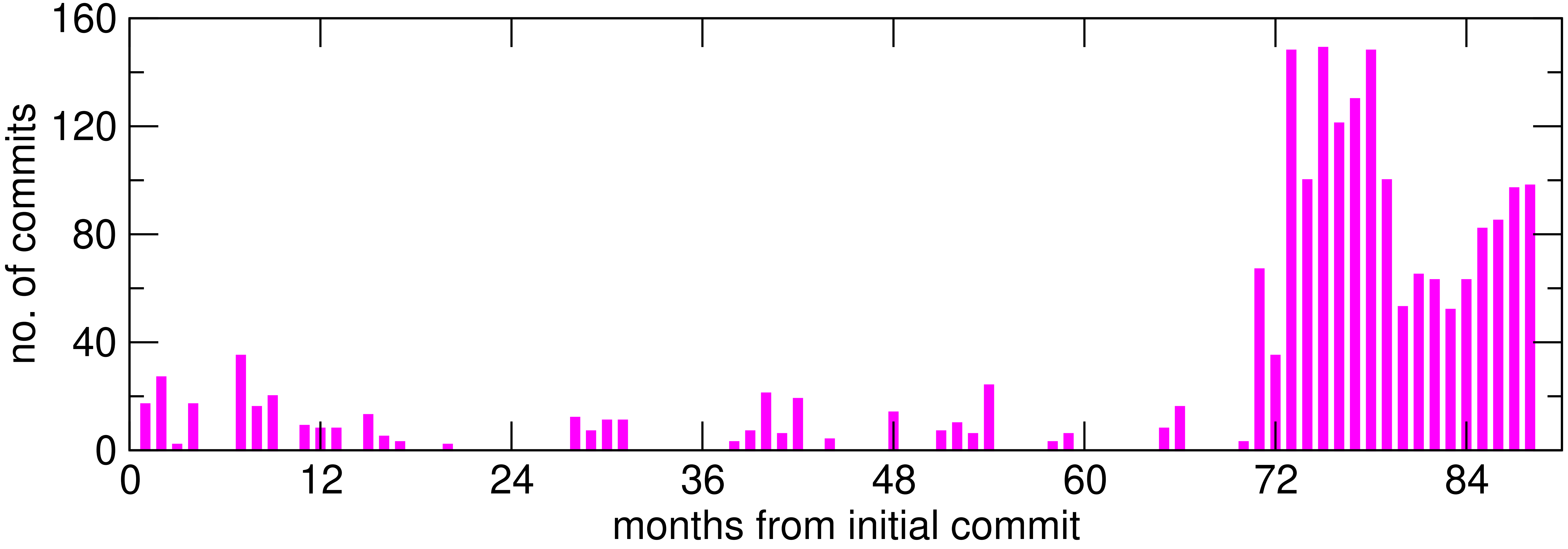
- **1. June 2017:** Version 0.9a1: Initial public release (alpha).
- **22. June 2017:** 2nd public release (alpha).
- **30. April 2019:** Version 1.0: 3rd public release.
- **02. March 2021:** Version 1.1: 4th public release.
- **22. March 2023:** Add README.md, CONTRIBUTING, CODE_OF_CONDUCT.md (no new version, no CHANGELOG)

sqaaas software bronze

EOSC-SYNERGY · SOFTWARE · COMPLIANT

Working repository

- **41 branches**, many experimental, some dead
- Roughly, 5 categories:
  - Structural changes
  - Observables
  - Exploration of particular techniques (e.g. noise reduction)
  - GPU porting
  - CI/CD



Working repository ===> Public repository

Core code vs. observables

- Priorities for core code:
  - **Long-term maintenance**
    The permanent members of the collaboration must understand the code deeply at all times (for long-term planning and newcomers training).
  - **Keep some level of compatibility with openQCD**
    Keep open the possibility to import new features of openQCD with a relatively minimal efforts. However, it is not clear this is still possible.
- Result: changes to the core code are very slow. Still, there are many things that could be improved
  e.g. refactor event and parameter database, reduce module interdependence, increase safety, design general interfaces (e.g. for solvers), refactor input file parsing and validation, design a strategy for automating testing...
- Refactoring of the core code must be done in a holistic way and it would take the focus away from physics. *Not for now...*

Working repository ===> Public repository

Core code vs. observables

- Next release should include:
    - Fix of various compiler warnings (e.g. concerning string length)
    - Add autoappend feature to ms* programs (i.e. infer the initial configuration from *.log and *.dat files)
    - Add calculation of the Pfaffian sign (ms7)
    - Add calculation of mass reweighting factor

Working repository ===> Public repository

Core code vs. observables

- So far, no observables have been published. This will change in the future...
  - More flexibility, we can explore code structures that depart from the core code. We can allow for C99, linking with external libraries...
  - Still, observables should be fully tested (this is difficult!) and thoroughly documented.
  - Coordination is needed: some features are needed in several observables and they should be agreed upon.
  - Write observable code in such a way that it is easy to use GPU solvers. **We need to agree on the interface!**
- This is something we should invest on.

Working repository ===> Public repository

Core code vs. observables

- Short term
    - RM123 insertions with frequency splitting, hopping expansion and, perhaps, low-mode averaging
    - Electromagnetic current 2pt function (connected and disconnected)
    - π0 two-point functions
- Medium term
    - Baryons (with all Wick contractions) with smeared sources
- Long term
    - ...

Keep observables separated and organized

```
openQxD/
   ➡ devel/
   ➡ doc/
   ➡ extras/
      ➡ devel/
         ➡ lowrnk/
         ➡ msrw/
      ➡ doc/
      ➡ include/
      ➡ main/
         ➡ lowrnk/
         ➡ msrw/
      ➡ modules/
         ➡ invdir/
         ➡ lowrnk/
         ➡ msrw/
   ➡ include/
   ➡ main/
   ➡ minmax/
   ➡ modules/
```

`invdir`

Generic functions to read and write solver-related parameters, apply solvers and collect statistics. QUDA interface should be buried here.

`lowrnk`

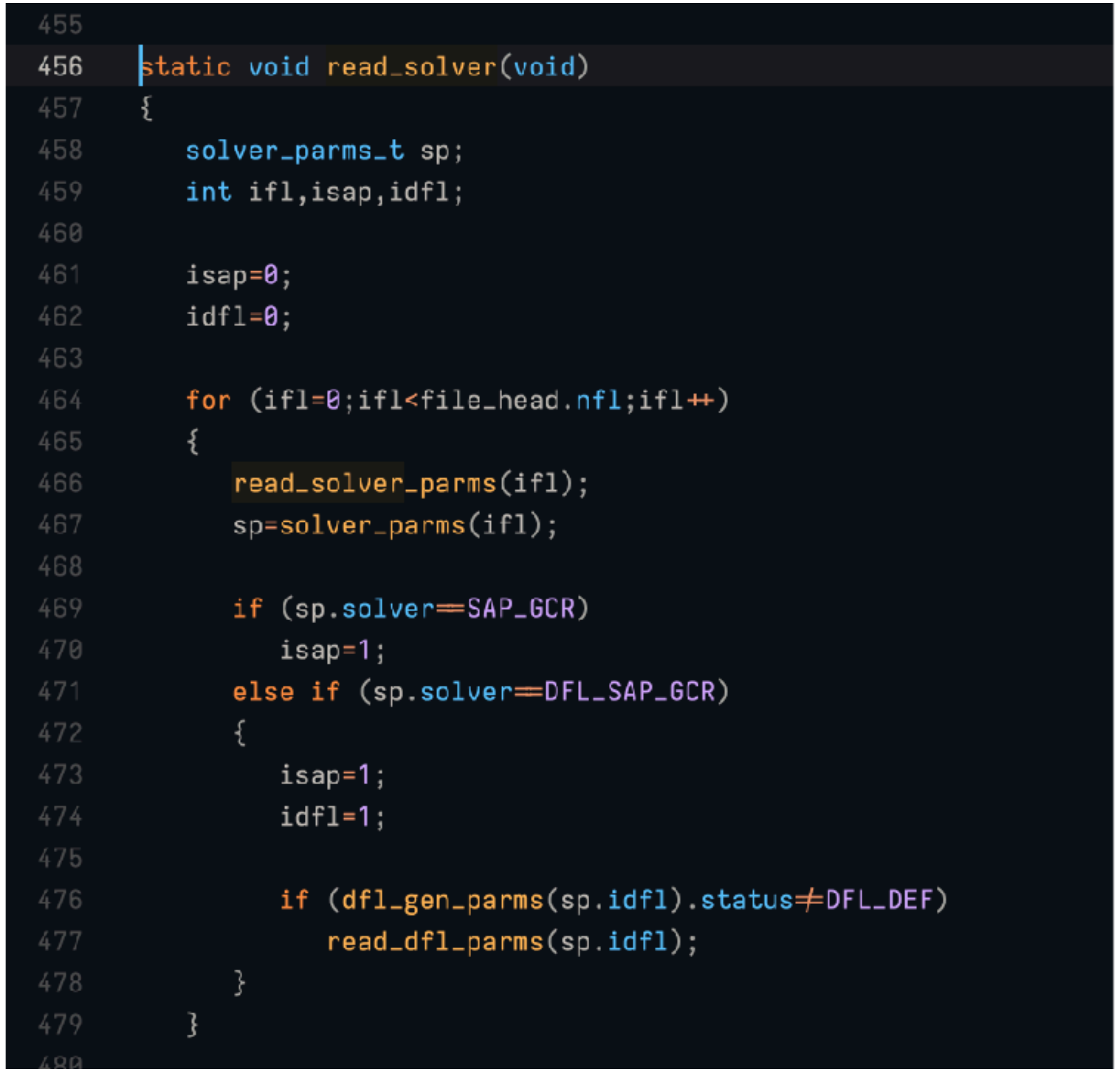
Abstract interface for various noise-reduction techniques based on low-rank approximations of the inverse of the Dirac operator. Currently implemented: frequency-splitting, hopping expansion, rough-solver approximation. Low-mode averaging should be added here.

`msrw`

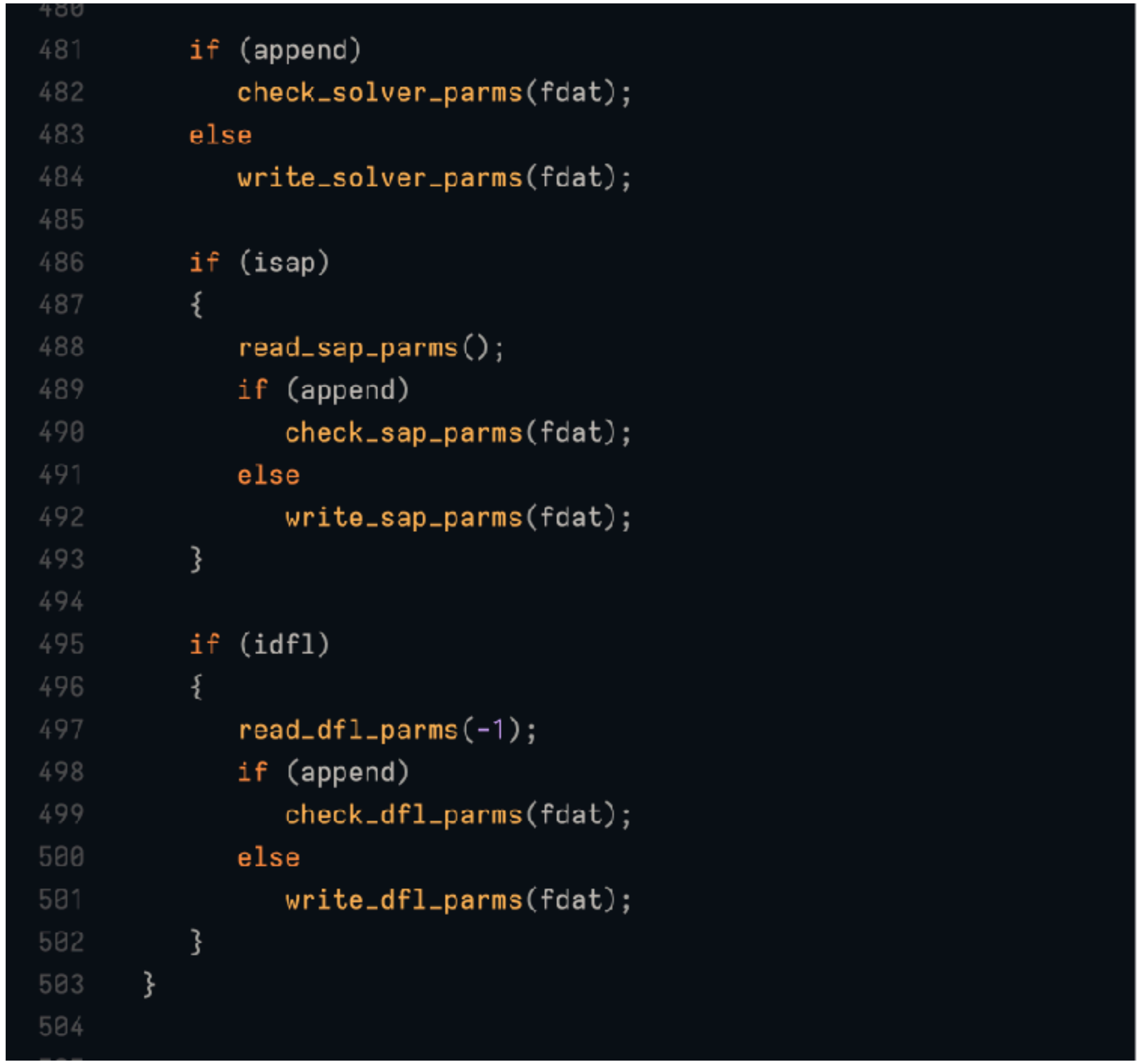
Calculation of mass-reweighting factor.

Using solvers (e.g. ms6)

- Executed once: **Read input parameters (for solvers, SAP, deflation).**



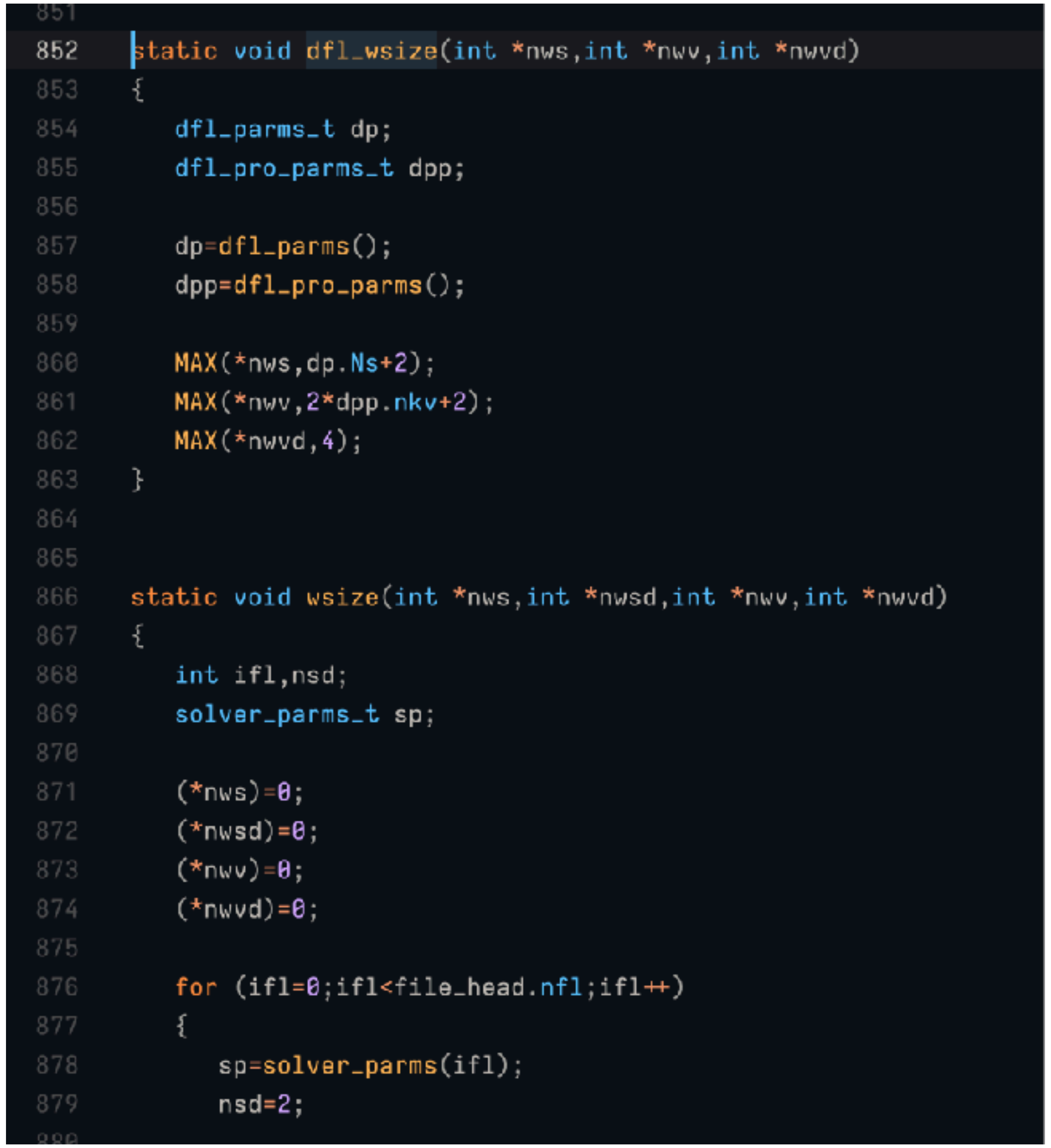
```c
static void read_solver(void)
{
   solver_parms_t sp;
   int ifl,isap,idfl;

   isap=0;
   idfl=0;

   for (ifl=0;ifl<file_head.nfl;ifl++)
   {
      read_solver_parms(ifl);
      sp=solver_parms(ifl);

      if (sp.solver==SAP_GCR)
         isap=1;
      else if (sp.solver==DFL_SAP_GCR)
      {
         isap=1;
         idfl=1;

         if (dfl_gen_parms(sp.idfl).status!=DFL_DEF)
            read_dfl_parms(sp.idfl);
      }
   }
```



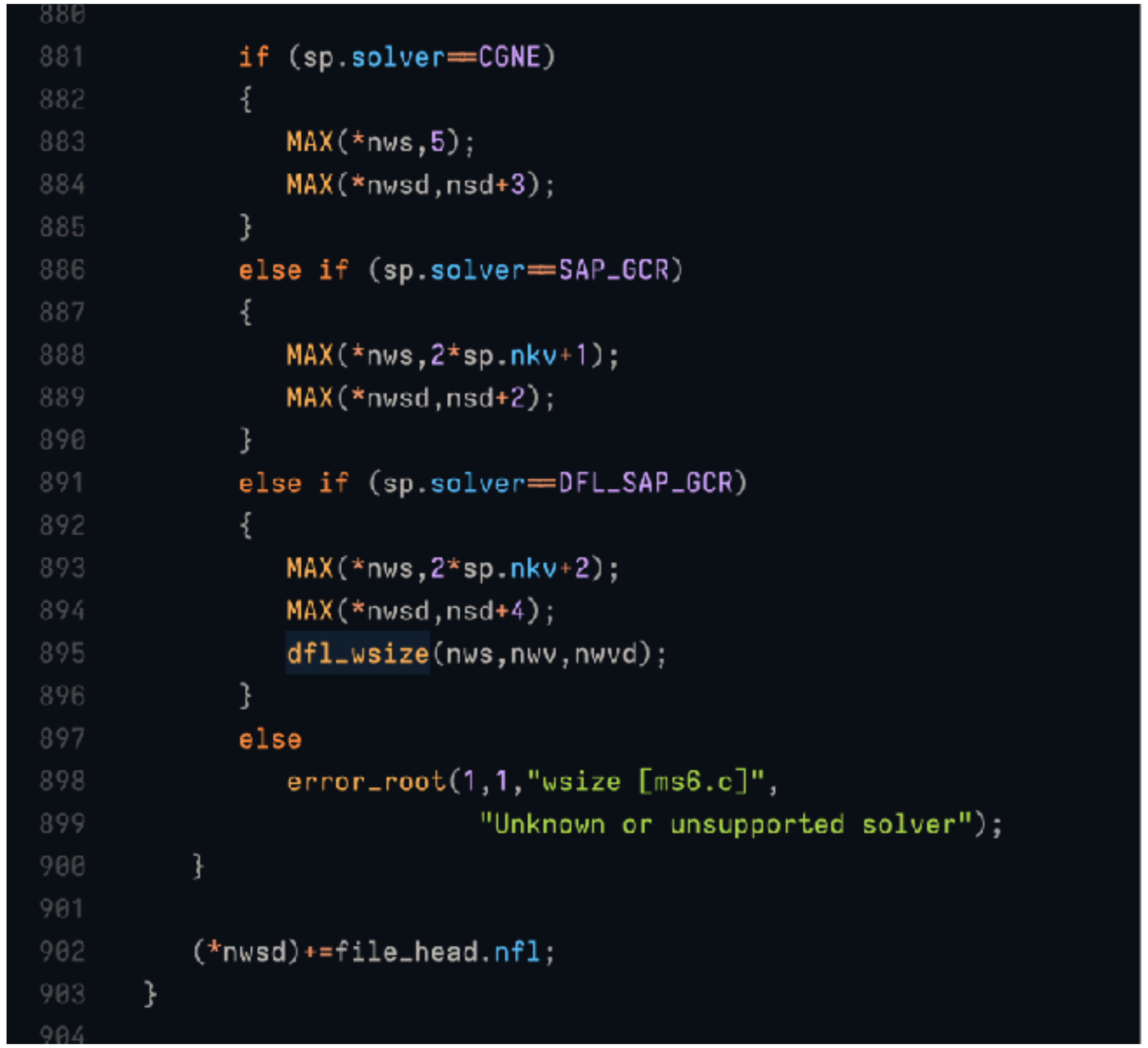
```c
   if (append)
      check_solver_parms(fdat);
   else
      write_solver_parms(fdat);

   if (isap)
   {
      read_sap_parms();
      if (append)
         check_sap_parms(fdat);
      else
         write_sap_parms(fdat);
   }

   if (idfl)
   {
      read_dfl_parms(-1);
      if (append)
         check_dfl_parms(fdat);
      else
         write_dfl_parms(fdat);
   }
}
```

Using solvers (e.g. ms6)

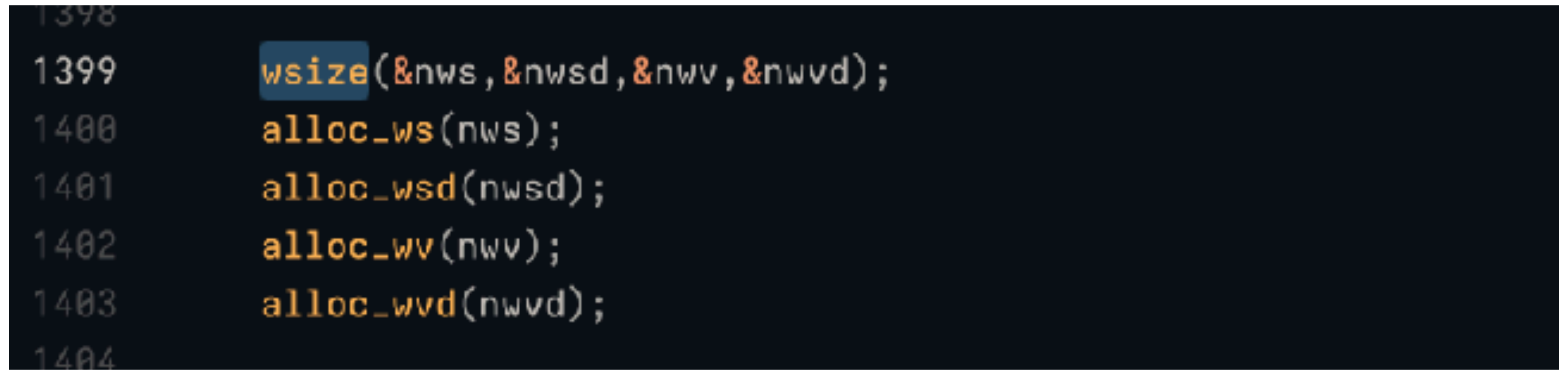
- Executed once: **Calculate and allocate the needed workspaces.**



```
static void dfl_wsize(int *nws,int *nwv,int *nwvd)
{
   dfl_parms_t dp;
   dfl_pro_parms_t dpp;

   dp=dfl_parms();
   dpp=dfl_pro_parms();

   MAX(*nws,dp.Ns+2);
   MAX(*nwv,2*dpp.nkv+2);
   MAX(*nwvd,4);
}


static void wsize(int *nws,int *nwsd,int *nwv,int *nwvd)
{
   int ifl,nsd;
   solver_parms_t sp;

   (*nws)=0;
   (*nwsd)=0;
   (*nwv)=0;
   (*nwvd)=0;

   for (ifl=0;ifl<file_head.nfl;ifl++)
   {
      sp=solver_parms(ifl);
      nsd=2;
```



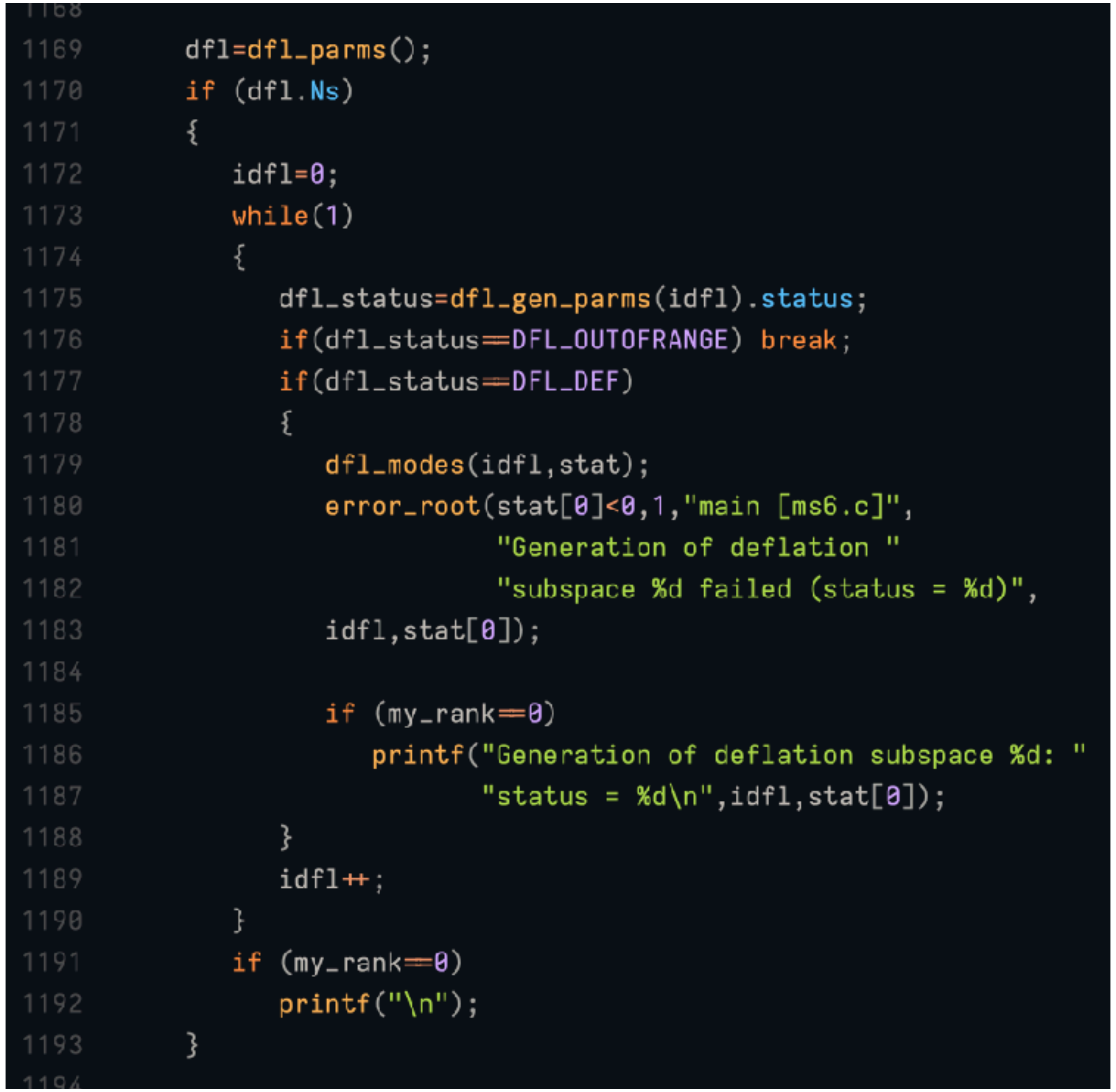
```
      if (sp.solver==CGNE)
      {
         MAX(*nws,5);
         MAX(*nwsd,nsd+3);
      }
      else if (sp.solver==SAP_GCR)
      {
         MAX(*nws,2*sp.nkv+1);
         MAX(*nwsd,nsd+2);
      }
      else if (sp.solver==DFL_SAP_GCR)
      {
         MAX(*nws,2*sp.nkv+2);
         MAX(*nwsd,nsd+4);
         dfl_wsize(nws,nwv,nwvd);
      }
      else
         error_root(1,1,"wsize [ms6.c]",
                    "Unknown or unsupported solver");
   }

   (*nwsd)+=file_head.nfl;
}
```



```
   wsize(&nws,&nwsd,&nwv,&nwvd);
   alloc_ws(nws);
   alloc_wsd(nwsd);
   alloc_wv(nwv);
   alloc_wvd(nwvd);
```

Using solvers (e.g. ms6)

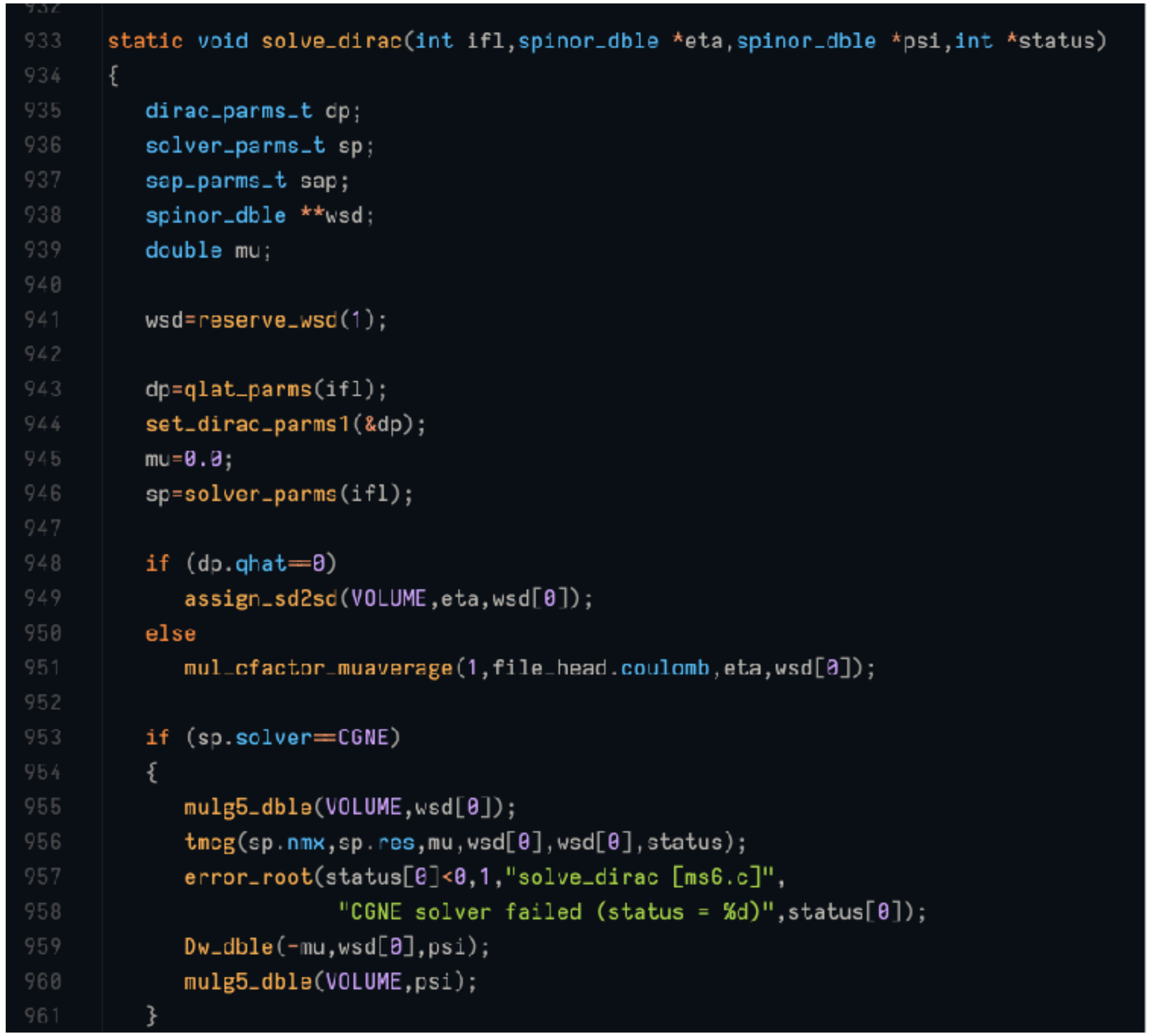
- Executed every time a gauge configuration is read: **Calculate deflation subspaces.**



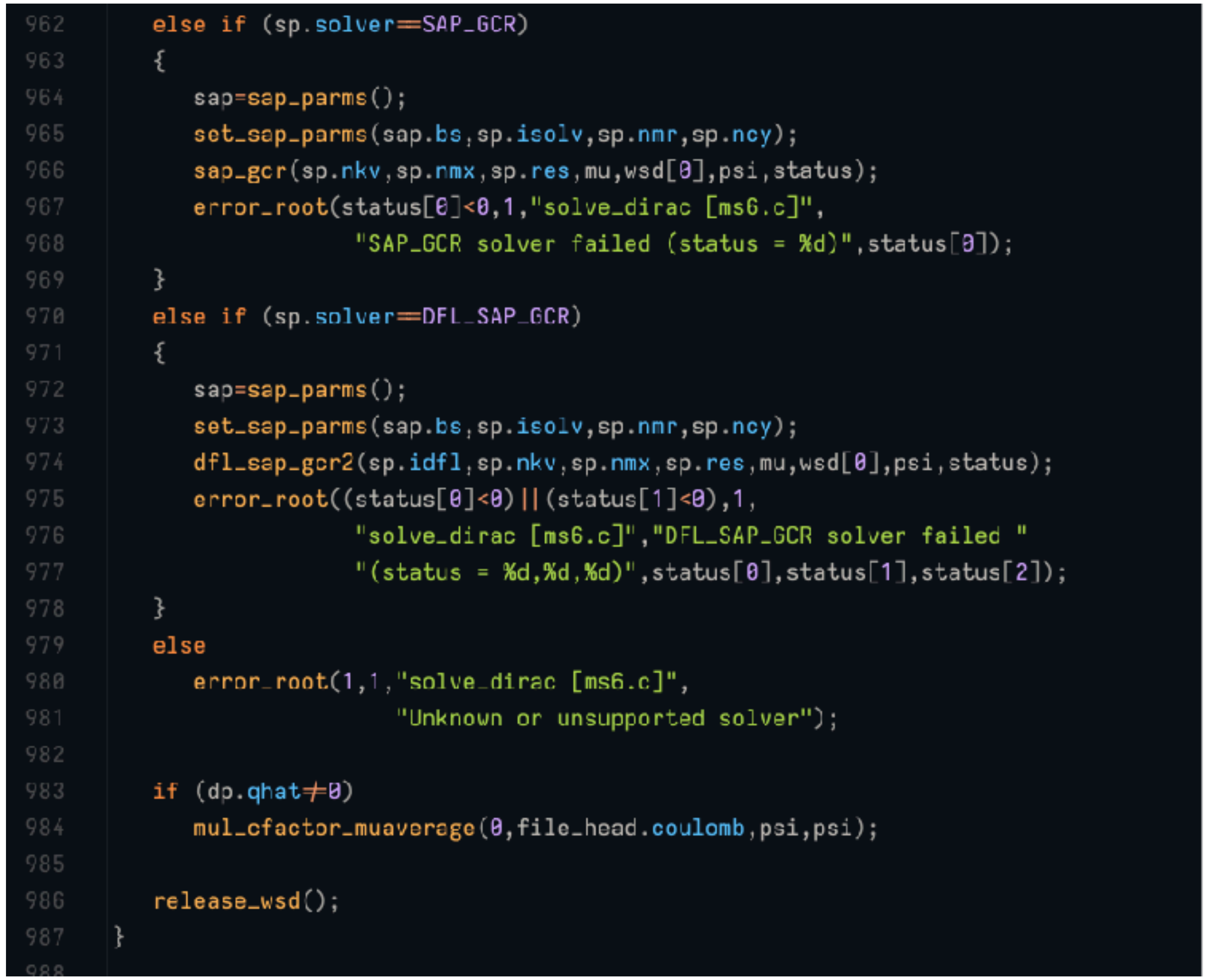
```
   dfl=dfl_parms();
   if (dfl.Ns)
   {
      idfl=0;
      while(1)
      {
         dfl_status=dfl_gen_parms(idfl).status;
         if(dfl_status==DFL_OUTOFRANGE) break;
         if(dfl_status==DFL_DEF)
         {
            dfl_modes(idfl,stat);
            error_root(stat[0]<0,1,"main [ms6.c]",
                       "Generation of deflation "
                       "subspace %d failed (status = %d)",
            idfl,stat[0]);

            if (my_rank==0)
               printf("Generation of deflation subspace %d: "
                      "status = %d\n",idfl,stat[0]);
         }
         idfl++;
      }
      if (my_rank==0)
         printf("\n");
   }
```

Using solvers (e.g. ms6)

- Executed every time we need to invert the Dirac operator: **Call solvers.**



```
static void solve_dirac(int ifl,spinor_dble *eta,spinor_dble *psi,int *status)
{
   dirac_parms_t dp;
   solver_parms_t sp;
   sap_parms_t sap;
   spinor_dble **wsd;
   double mu;

   wsd=reserve_wsd(1);

   dp=qlat_parms(ifl);
   set_dirac_parms1(&dp);
   mu=0.0;
   sp=solver_parms(ifl);

   if (dp.qhat==0)
      assign_sd2sd(VOLUME,eta,wsd[0]);
   else
      mul_cfactor_muaverage(1,file_head.coulomb,eta,wsd[0]);

   if (sp.solver==CGNE)
   {
      mulg5_dble(VOLUME,wsd[0]);
      tmcg(sp.nmx,sp.res,mu,wsd[0],wsd[0],status);
      error_root(status[0]<0,1,"solve_dirac [ms6.c]",
                 "CGNE solver failed (status = %d)",status[0]);
      Dw_dble(-mu,wsd[0],psi);
      mulg5_dble(VOLUME,psi);
   }
```



```
   else if (sp.solver==SAP_GCR)
   {
      sap=sap_parms();
      set_sap_parms(sap.bs,sp.isolv,sp.nmr,sp.ncy);
      sap_gcr(sp.nkv,sp.nmx,sp.res,mu,wsd[0],psi,status);
      error_root(status[0]<0,1,"solve_dirac [ms6.c]",
                 "SAP_GCR solver failed (status = %d)",status[0]);
   }
   else if (sp.solver==DFL_SAP_GCR)
   {
      sap=sap_parms();
      set_sap_parms(sap.bs,sp.isolv,sp.nmr,sp.ncy);
      dfl_sap_gcr2(sp.idfl,sp.nkv,sp.nmx,sp.res,mu,wsd[0],psi,status);
      error_root((status[0]<0)||(status[1]<0),1,
                 "solve_dirac [ms6.c]","DFL_SAP_GCR solver failed "
                 "(status = %d,%d,%d)",status[0],status[1],status[2]);
   }
   else
      error_root(1,1,"solve_dirac [ms6.c]",
                    "Unknown or unsupported solver");

   if (dp.qhat!=0)
      mul_cfactor_muaverage(0,file_head.coulomb,psi,psi);

   release_wsd();
}
```

Using solvers (just an example)

- Executed once: **Read input parameters (for solvers, SAP, deflation).**

  Introduce function which reads all relevant parameters, if they have not been read yet.

  ```
  void read_solver_sap_dfl_parms(int isp);
  ```

- Executed once: **Calculate and allocate the needed workspaces.**

  Introduce function that calculates workspace needed for all solvers

  ```
  void solver_and_dfl_wsize(int *nwud,int *nwad,int *nws,
                            int *nwsd,int *nwv,int *nwvd);
  ```

- Executed every time a gauge configuration is read: **Calculate deflation subspaces.**

  Remove this, and decide whether to calculate the deflation subspace based on event database.

- Executed every time we need to invert the Dirac operator: **Call solvers.**

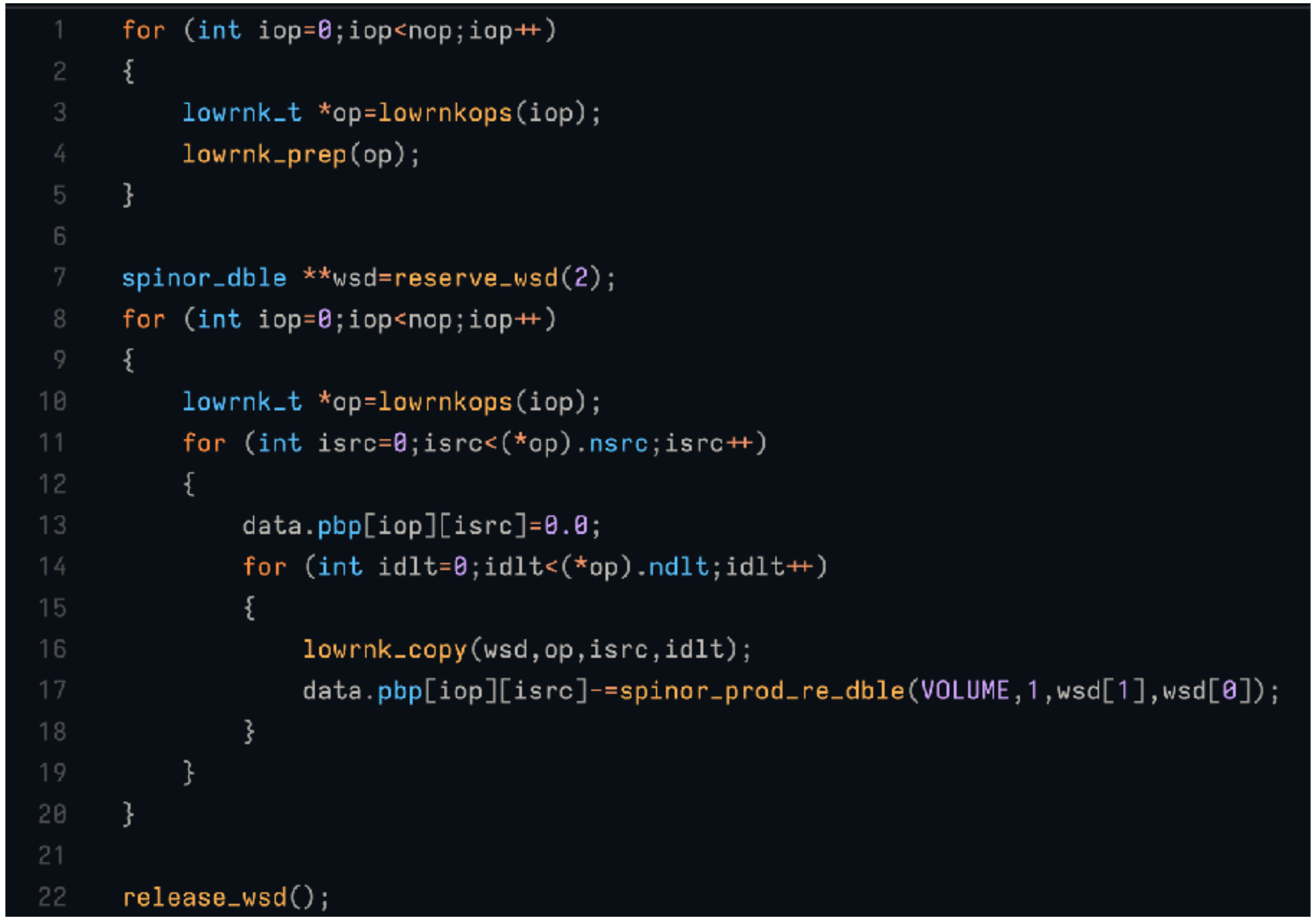
  Introduce function that calculates deflation subspace if necessary, initialize in vector to zero if required, invert Dirac operator with given solver, return solver and deflation status array if `status!=NULL`, check result and returns residue.

  ```
  double Dinv(int isp,spinor_dble *in,spinor_dble *out,int init,int *status);
  ```

Low rank approximation of inv(D)

A large class of noise-reduction techniques can be represented as

$$D^{-1} = \sum_{A} \langle\langle O_A \rangle\rangle \qquad O_A = \frac{1}{N^A_{src}} \sum_{n=1}^{N^A_{src}} \sum_{k=1}^{N^A_{dlt}} \psi_{A,n,k} \eta^{\dagger}_{A,n,k}$$



```
for (int iop=0;iop<nop;iop++)
{
    lowrnk_t *op=lowrnkops(iop);
    lowrnk_prep(op);
}

spinor_dble **wsd=reserve_wsd(2);
for (int iop=0;iop<nop;iop++)
{
    lowrnk_t *op=lowrnkops(iop);
    for (int isrc=0;isrc<(*op).nsrc;isrc++)
    {
        data.pbp[iop][isrc]=0.0;
        for (int idlt=0;idlt<(*op).ndlt;idlt++)
        {
            lowrnk_copy(wsd,op,isrc,idlt);
            data.pbp[iop][isrc]-=spinor_prod_re_dble(VOLUME,1,wsd[1],wsd[0]);
        }
    }
}

release_wsd();
```

Calculation of tr(inv(D))

`op [pointer to an instance of a derived class of lowrnk_t (which is virtual)]`

Represents the particular noise-reduction technique.

`lowrnk_prep [polymorphic function]`

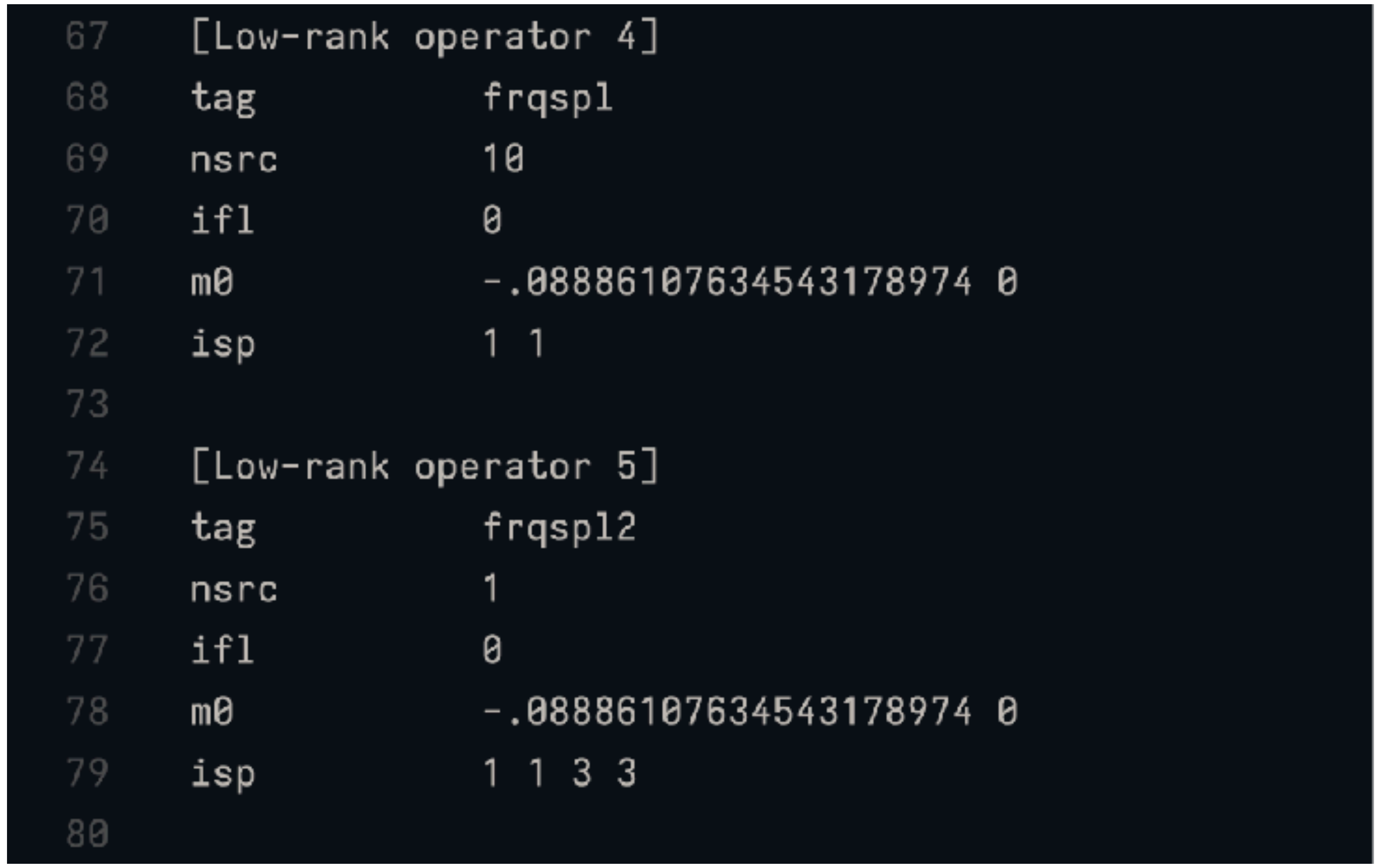
Calculates the psi and eta pseudofermions.

`lowrnk_copy [polymorphic function]`

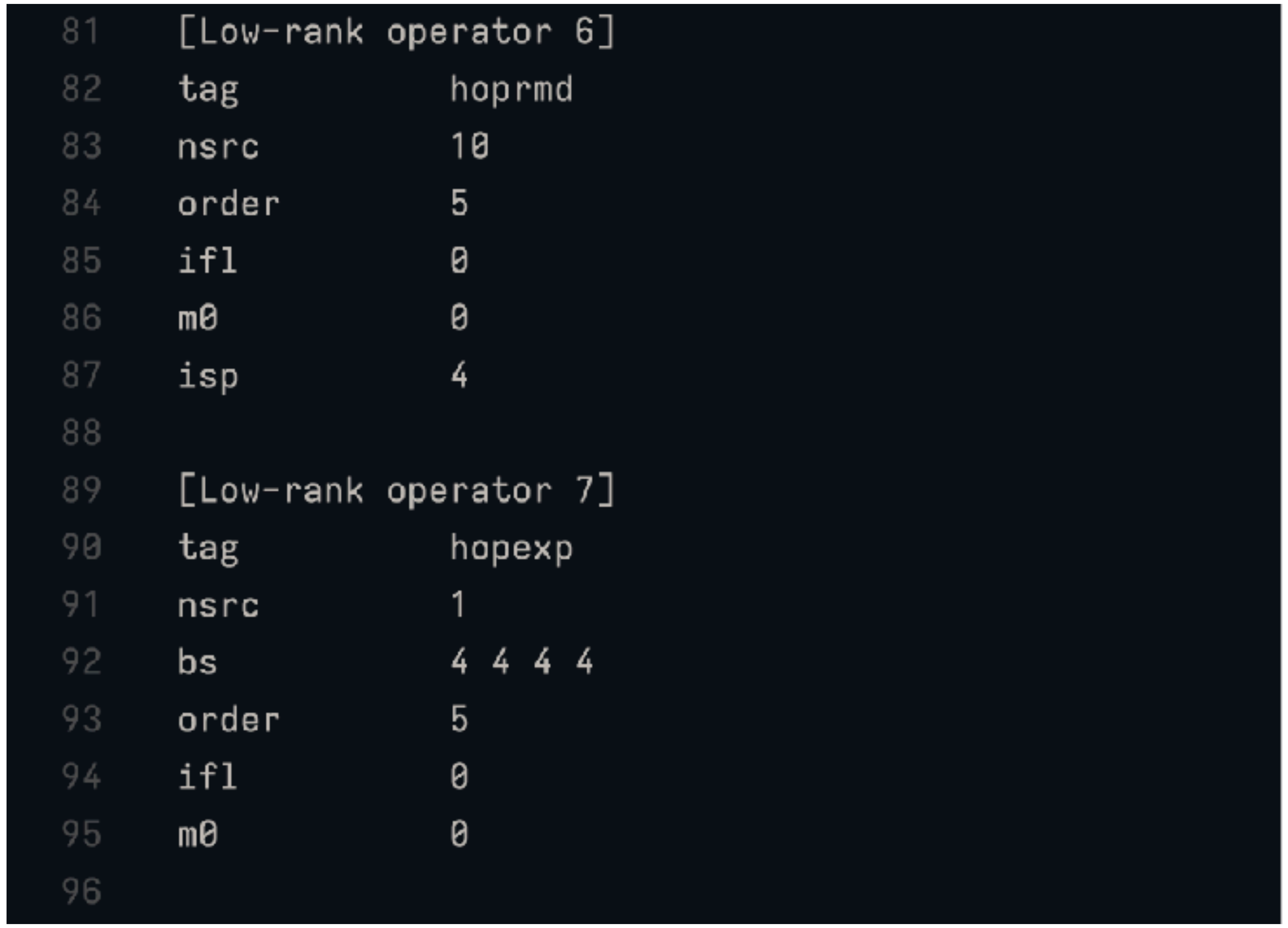
Copies the psi and eta pseudofermions for use.

Low rank approximation of inv(D)

In this case, the low-rank approximation is defined in the input file:



```
[Low-rank operator 4]
tag          frqspl
nsrc         10
ifl          0
m0           -.08886107634543178974 0
isp          1 1

[Low-rank operator 5]
tag          frqspl2
nsrc         1
ifl          0
m0           -.08886107634543178974 0
isp          1 1 3 3

```



```
[Low-rank operator 6]
tag          hoprmd
nsrc         10
order        5
ifl          0
m0           0
isp          4

[Low-rank operator 7]
tag          hopexp
nsrc         1
bs           4 4 4 4
order        5
ifl          0
m0           0

```

Low rank approximation of inv(D)

```
openQxD/
  ➡ [...]
  ➡ extras/
      ➡ [...]
      ➡ include/
          ➡ lowrnk.h
      ➡ modules/
          ➡ lowrnk/
              ➡ lowrnk.c
              ➡ lowrnk_database.c
              ➡ lowrnk_frqspl.c
              ➡ lowrnk_frqspl2.c
              ➡ lowrnk_hopexp.c
              ➡ lowrnk_hoprmd.c
              ➡ lowrnk_invdop.c
              ➡ lowrnk_invdop2.c
```

Expandable by adding independent pieces of code, without meddling with existing code!

- Add 4 lines to `lowrnk.h`
- Add one file in `modules/lowrnk` (following the same structure of all others)
- Add the new file to the Makefile
- When a new noise-reduction technique is added, nothing needs to be changed in main programs!