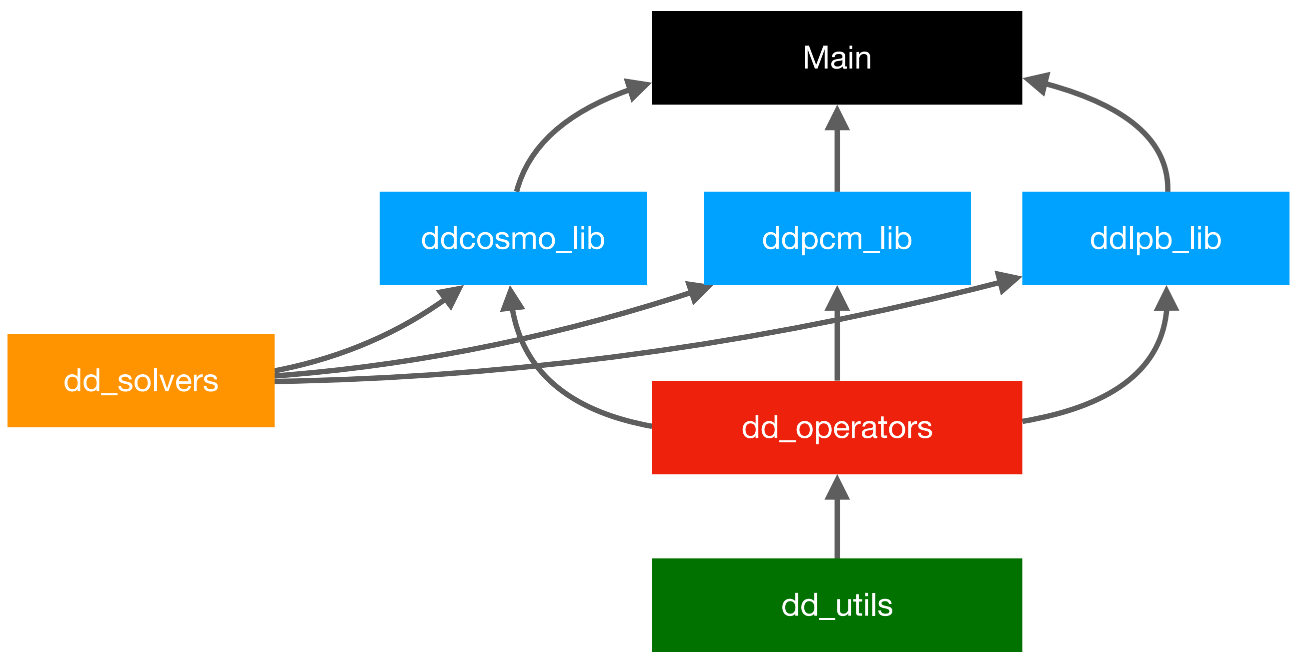
ddX

Modules:

- ddcosmo_lib
- ddpcm_lib
- ddlpb_lib
- dd_operators
- dd_solvers
- dd_utlis



Main

Filename:	Main.f90
Use:	ddcosmo_lib, ddpcm_lib, ddlpb_lib

Comments:

- The general idea is that this is a minimal (but sufficient) interface to couple ddX with an external driver routine (QM-code, Tinker, PSI4, …)
- The final computation to the (solvent) Fock-matrix can be done in ddX or by the mother driver
- We keep the existing structure with a few exceptions:
  - The implementation of the subroutines **ddinit** and **ddfree** (former memfree) are moved dd_utils. This is the minimal interface with an enduser.

- In variables (optional): epsin, kappa, eta, tol, lmax, ngrid, se

ddinit (in variables):

- qm: ddx is coupled to a QM-code or not
- (if qm=1) contract_qm: contract the DFT-integrals within ddX or not
- do_force: compute force or not
- (if qm=1) do_fock: compute the contribution to the Fock matrix in ddx or not
- (if qm=1 “and” (do_fock=1 “or” do_force=1)) ndftquad, dftquad_pt: DFT quadrature points
- (if qm=1 “and” contract_qm=1) dftquad_wght, grad_dftquad_wght: DFT quadrature weights and gradient of weights
- nsph, csph, rsph, epsout (see ddutils)
- (optional): epsin, kappa, eta, tol, lmax, ngrid, se (see ddutils)

ddinit (out variables):

- ncav, ccav

ddx (in variables):

- charge: nuclear point-charges
- (if qm=1 “and” (do_fock=1 “or” do_force=1)) rho: el. density at DFT quadrature points
- phi: potential at Lebedev points ccav
- (if do_force=1 “or” do_fock=1) gradphi, hessphi: field and field-gradient at Lebedev points ccav
- (if qm=1 “and” contract_qm=0) psi: psi-vector corresponding to rho (el. density)
- (if qm=1 “and” contract_qm=1) psimunu, phimunu

ddx (out variables):

- esolv
- (if do_force=1) fx
- (if qm=1 “and” do_fock=1 “and” contract_qm=1) focksolv
- (if qm=1 “and” do_fock=1 “and” contract_qm=0) etajg, xinj

Content of main:

>> call ddinit(model, nsph, csph, rsph, epsout, qm, contract_qm, do_force, do_fock, ndftquad, dftquad_pt, dftquad_wght, grad_dftquad_wght, + other optionals)
>> compute phi(ccav)
>> if(qm=1 “and” contract_qm=0): compute psi
>> if(qm=1 “and” (do_fock=1 “or” do_force=1)): compute rho(dftquad_pt)
>> call ddx(phi, gradphi, hessphi, psi, gradpsi, psimunu, phimunu, rho, charge, esolv, fx, focksolv, etajg, zetanj)
>> (if qm=1 “and” contract_qm=0 “and” do_fock=1): compute contribution to Fock-matrix here: Eqns (22,25/28) of [1]
>> (if qm=1 “and” contract_qm=0 “and” do_force=1): compute QM-contribution to the gradient: Eqns (47/48,50) of [2]

```
>>  call ddfree(…)
```

Module name: ddcosmo_lib

File name: ddcosmo.f90

Use: dd_solvers, dd_operators, dd_utils

Variables:
- X
- S
- g

Subroutines:
- ddcosmo: solve primal, compute energy, compute adjoint and force resp. Fock matrix if required:
  - call [g,Psi] = mkrhs(-Phi,rho): get rhs for primal and dual linear system
  - call X = solver(Lx,g): solve primal linear system
  - esolv = 0.5*f(epsout)*sprod(Psi,X)
  - if(do_fock or do_force)
    - call S = solver(Lstarx,psi)
  - endif
  - if(do_fock>0)
    - compute xinj: call xi = ddeval(-S, at exterior points only with U_i weights)
    - compute etajg
  - if(do_fock=1)
    - Focksolv = 0.5*f(epsout)*(sprod(etajg*dftquad_wght,psimunu) + sprod(xinj,phimunu))
  - endif
  - if(do_force)
    - … (too complicated to be reported here!)
  - endif

Module name: ddpcm_lib

File name: ddpcm.f90

Use: dd_solvers, dd_operators, dd_utils

Variables:
- X, Phie
- S, Y, Q
- g (or Phi)

Subroutines:
- ddpcm: solve primal, compute energy, compute adjoint and force resp. Fock matrix if required:
  - call g = mkrhs(Phi,rho): get rhs for primal and dual linear system
  - call rhs = Rinfx(g): prepare rhs
  - call Phie = solver(Repsx,rhs): solve first primal linear system
  - call X = solver(Lx,-Phie): solve second primal linear system
  - esolv = 0.5*sprod(psi,X)
  - if(do_fock or do_force)
    - call S = solver(Lstarx,Psi)
    - call Y = solver(Repsstarx, S)
    - Q = S-4*pi*frac{1}{epsout - 1}*Y
  - endif
  - if(do_fock)
    - call xi = ddeval(Q, at exterior points only with U_i weights -> check if this is true)
    - Focksolv = 0.5*(sprod(psimunu,X) + sprod(xi,phimunu))

        - endif
        - if(do_force)
            - … (too complicated to be reported here!)
        - endif

Module name: dd_operators

File name: dd_operators.f90

Use: dd_utils

Variables:

Subroutines:
- Lx, Lstarx, Ldm1x: Cosmo matrix, adjoint matrix, and diagonal preconditioner mat-vec operations
- D, Dstarx: global double layer operator and adjoint mat-vec
- Repsx, Repsstarx:
- Lkappax: HSP-mat-vec product with dd-strategy
- Sx, Sstarx: global single layer operator mat-vec
- Skappax, Skappastarx: global single layer operator mat-vec for LPB
- DtNx: local DtN mat-vec
- DtNkappax: local DtN mat-vec for LPB
- g (assemble rhs with U_i weight)
- gradL (former fdoka + fdokb)
- gradg (former fdoga)
- gradR
- gradS, gradSkappa, gradLkappa

Module name: dd_solvers

File name: dd_solvers.f90

Use: none

Variables:
- todo

Subroutines:
- gmres: main argument: matvec (specified in ddcosmo_lib, ddpcm_lib, ddlpb_lib)
- diis: main arguments: preconx, matvec (specified in ddcosmo_lib, ddpcm_lib, ddlpb_lib)

Module name: dd_utils

File name: dd_utils.f90

Use: none

Variables:
- model
- csph
- rsph
- nsph
- charge
- epsin, epsout
- kappa
- eta
- tol

- lmax
- ngrid
- shift (se)
- ndftquad,dftquad_pt,dftquad_wght

Subroutines:

- ddinit
- ddfree (former memfree)
- sprod, fsw, dfsw
- ptcart, prtsph
- ylmbas, dbasis, polleg, trgev
- wghpot
- hsnorm, hnorm
- header
- calcv (required in Lx)
- adjrhs (required in Lstarx)
- intlmp (required in …)
- intrhs -> will be replaced
- ddmkxi (required in …) -> will be replaced
- ddproject (all, only interior, only exterior: from nodal to modal) -> replacement for intrhs
- ddeval (evaluation SH's series at integration points, + multiply optionally by U_i) -> replacement for ddmkxi
- [g,Psi] = **mkrhs**(Phi,rho,Z)
    - g = ddproject(-Phi)
    - if(do_force || do_fock)
        - Psi = assemble Psi-vector according to Eqn (20/21) IJQC-paper

`[1]:` B. Stamm, L. Lagardère, G. Scalmani, P. Gatto, E. Cancès, J.-P. Piquemal, Y. Maday, B. Mennucci, F. Lipparini, How to make continuum solvation incredibly fast in a few simple steps: a practical guide to the domain decomposition paradigm for the Conductor-like Screening Model, Int. J. Quantum Chem., (2018)

`[2]:` F. Lipparini, G. Scalmani, L. Lagardère, B. Stamm, E. Cancès, Y. Maday, J.-P. Piquemal, M. Frisch and B. Mennucci, Quantum, Classical and Hybrid QM/MM Calculations in Solution: General Implementation of the ddCOSMO Linear Scaling Strategy, J. Chem. Phys., Vol. 141, pp. 184108 (2014)