User's Manual for HODIF, A Library of High-Order Derivatives, Interpolations and Filters

Jaideep Ray
Sandia National Laboratories, Livermore, CA

April, 2006

Abstract

This User's Manual describes how one may use HODIF, a library of High-Order derivatives, interpolations and filters. This document is relevant to Version 0.1 of HODIF, which addresses data stored at cell-corners (also known as vertex-centered collocation) of a discretized domain. The mesh is assumed to be uniform and Cartesian. The library is designed to be used in constructing high-order versions of block-structured adaptive meshes.

1 Introduction

HODIF is a library of high-order derivatives, interpolations and filters. The current version, Version 0.1 , is restricted to vertex-centered variables i.e. where variables are stored at cell-corners of a uniformly discretized domain. Further, the mesh is assumed to be Cartesian. The current version can only handle 1D and 2D meshes; 3D is in the works. The library is designed to work within the context of high-order block-structured adaptive mesh (AMR) simulations, where regions are refined in rectangular patches - see [2] for details. Filters are an integral part of block-structured AMR simulations, especially when Runge phenomenon is observed.

HODIF is written in FORTRAN77, but like most other FORTRAN77 libraries, can be invoked from C/C++. However, one needs to specify both the FORTRAN and C/C++ compilers being used in order to deal with multi-language interoperability issues. This is explained in Sec. 2.

New versions of HODIF will be announced at [1].

2 Structure of the package, compiling and installing

The HODIF package consists of three directories `deriv/`, `interp/` and `filter/` which is where the derivatives, interpolations and filters are kept. Each has a Makefile and can compile them into `libDerivatives.a`, `libFilters.a` and `libVertexCenteredInterps.a`. These are copied to `lib/` which is the correct "final resting place" for the libraries.

`include/` contains `deriv.h, interp.h` and `filter.h`. These are header files, in case you wish to use the libraries from C/C++. Examples of how you may use the library from FORTRAN or C/C++ are in `example/`. This user's manual is in `doc/`.

The compilation of the package is easy. In the top directory, edit the file `MakeIncl.HODIF`. It will require the specification of certain compilers and flags. Doing `make` should create the libraries in `lib/`. Link them in, as per the `example/Makefile`.

3 Description of the library

3.1 Filter

`libFilters.a` has only one function in it. For C/C++, the interface is in `include/filter.h`

```
FORTRAN :

  subroutine expfil(fn, ifilt_x, ifilt_y, ifilt_z, iord,
                   fiL, fiR, fjL, fjR, fkL, fkR,
    piL, piR, pjL, pjR, pkL, pkR,
    wiL, wiR, wjL, wjR, wkL, wkR,
    ierror, error, ierr_coeff, err_coeff)

C/C++ :

 void C_expfil(double *fn, int *ifilt_x, int *ifilt_y, int *ifilt_z, int *iord,
               int *fiL, int *fiR, int *fjL, int *fjR, int *fkL, int *fkR,
               int *piL, int *piR, int *pjL, int *pjR, int *pkL, int *pkR,
               int *wiL, int *wiR, int *wjL, int *wjR, int *wkL, int *wkR,
               int *ierror, char *error, int *ierr_coeff, char *err_coeff) ;
```

where

1. `fn` is the 3D `real*8` INPUT FORTRAN array containing the solution to be filtered. It will be returned in the same array
2. `ifilt_x, ifilt_y, ifilt_z` are integers which denote if we should filter in x-, y- or z-directions. Setting them to 1 indicates "Yes", 0 to "No".
3. `iord`, an integer, denoting the order of the filter. Positive even numbers between 2 and 50 are OK.
4. `fiL, fiR, fjL, fjR, fkL, fkR` are integers which denote the lower and upper bounds to the 3D FORTRAN array `fn`.
5. `piL, piR, pjL, pjR, pkL, pkR` are integers which denote the lower and upper bounds to the 3D FORTRAN array `fn` where you actually have data that can be filtered. This is necessarily a subset of the array denoted by `fiL, fiR, fjL, fjR, fkL, fkR`.
6. `wiL, wiR, wjL, wjR, wkL, wkR` the lower and upper bounds of the array section of `fn` where you wish filtering to be done. Again this is a subset of `piL, piR, pjL, pjR, pkL, pkR`.
7. `ierror`, an integer error code. 0 means "No Error". -30 means that the filter stencil width is wider than the array supplied.
8. `error` a `character*60 error` which will contain the error in words. In C/C++, this should be `char error[256]`.

9. `ierr_coeff`, an integer error code. 0 means “No Error”. -10 means that you specified an odd-ordered filter stencil (we do not do that) and -20 indicates that a negative order has been indicated (all filters are positive, even ordered ones).

10. `err_coeff` a `character*60 error` which will contain the coefficients' error in words. In C/C++, this should be `char error[256]`.

3.2 Derivatives

`libDerivatives.a` has a number of functions. The function names end in `_co` indicating collation i.e. the derivatives are evaluated at grid points i.e at `(i, j)`. This distinction is made since one can also derive stencils which evaluate derivatives in a staggered manner i.e. at `(i+1/2, j+1/2)` .

```
FORTRAN :

  subroutine x_der1_co(f, df, dx, dy, dz,
                       vel, upwind, orderi, orderb,
                       biL, biR, bjL, bjR, bkL, bkR,
                       fiL, fiR, fjL, fjR, fkL, fkR,
                       iiL, iiR, ijL, ijR, ikL, ikR,
                       viL, viR, vjL, vjR, vkL, vkR,
                       dfiL, dfiR, dfjL, dfjR, dfkL,
                       dfkR, iperx, ipery, iperz, ierror,
                       char *error)

  subroutine y_der1_co( -- same argument sequence as x_der1_co -- )

  subroutine x_der2_co(f, df, dx, dy, dz,
                       orderi, orderb,
                       biL, biR, bjL, bjR, bkL, bkR,
                       fiL, fiR, fjL, fjR, fkL, fkR,
                       iiL, iiR, ijL, ijR, ikL, ikR,
                       dfiL, dfiR, dfjL, dfjR, dfkL,
                       dfkR, iperx, ipery, iperz, ierror,
                       char *error)

  subroutine y_der2_co( -- same argument sequence as x_der2_co -- ) ;

  subroutine x_der3_co( -- same argument sequence as x_der1_co -- )

  subroutine x_der4_co( -- same argument sequence as x_der1_co -- )

  subroutine x_der5_co( -- same argument sequence as x_der1_co -- )

C/C++ :

  void C_x_der1_co(double *f, double *df, double *dx, double *dy, double *dz,
                   double *vel, int *upwind, int *orderi, int *orderb,
                   int *biL, int *biR, int *bjL, int *bjR, int *bkL, int *bkR,
                   int *fiL, int *fiR, int *fjL, int *fjR, int *fkL, int *fkR,
                   int *iiL, int *iiR, int *ijL, int *ijR, int *ikL, int *ikR,
                   int *viL, int *viR, int *vjL, int *vjR, int *vkL, int *vkR,
                   int *dfiL, int *dfiR, int *dfjL, int *dfjR, int *dfkL,
                   int *dfkR, int *iperx, int *ipery, int *iperz, int *ierror,
                   char *error) ;

  void C_y_der1_co( -- same argument sequence as C_x_der1_co -- ) ;

  void C_x_der2_co(double *f, double *df, double *dx, double *dy, double *dz,
                   int *orderi, int *orderb,
                   int *biL, int *biR, int *bjL, int *bjR, int *bkL, int *bkR,
                   int *fiL, int *fiR, int *fjL, int *fjR, int *fkL, int *fkR,
                   int *iiL, int *iiR, int *ijL, int *ijR, int *ikL, int *ikR,
                   int *dfiL, int *dfiR, int *dfjL, int *dfjR, int *dfkL,
                   int *dfkR, int *iperx, int *ipery, int *iperz, int *ierror,
                   char *error) ;

  void C_y_der2_co( -- same argument sequence as C_x_der2_co -- ) ;

  void C_x_der3_co( -- same argument sequence as C_x_der1_co -- ) ;

  void C_x_der4_co( -- same argument sequence as C_x_der1_co -- ) ;

  void C_x_der5_co( -- same argument sequence as C_x_der1_co -- ) ;
```

The general format of the name of the subroutines is `D_derN_Y` where `D` is `x` or `y` indicating the direction of the derivative, `N` is `1-5` for first to fifth-derivative and `Y` is `co` for collocated collocation of the derivatives.

1. `f` is the 3D `real*8` INPUT FORTRAN array containing the field whose derivative is to be taken.
2. `df`, a 3D `real*8` OUTPUT FORTRAN array filled up by the subroutine with the derivatives.
3. `dx, dy, dz`, the mesh spacing in x-, y- and z-directions
4. `vel`, a velocity array, of the same size as `f` containing velocities at each `(i, j)`. This is used for upwinding first derivatives.
5. `upwind` an integer flag, which when set to 1 indicates "please upwind". Else set it to zero.
6. `orderi, orderb`, two integers that indicate the order of accuracy needed in the interior and at the boundary (`orderb`). `orderb` is currently neglected and the subroutines will close the stencil to a lower order by skewing the stencils. `orderi` can take positive, even values between 2 and 8 if not upwinding and between 3 and 7 if upwinding.
7. `biL, biR, bjL, bjR, bkL, bkR`, integers that denote the width of the "halo" around a rectangular domain. Each number describes the width at the lower and upper end of each axis.
8. `fiL, fiR, fjL, fjR, fkL, fkR`, integers which denote the lower and upper bounds of the indices of array `f`

9. `iiL, iiR, ijL, ijR, ikL, ikR`, integers which denote the lower and upper bounds of the section of array `f` where are the “interior” or “valid” points i.e. “non-halo” points.

10. `dfiL, dfiR, dfjL, dfjR, dfkL, dfkR`, integers which denote the lower and upper bounds of `df`.

11. `iperx, ipery, iperz` integers that denote if the data is periodic in x-, y- or z-directions. setting them to 1 means “Yes”, 0 is “No”. Anything else is an error.

12. `ierror`, an error return code. Zer means “OK”, else

    - -10 : the interior + halo exceed the size of the array
    - -20 : the interior is wider than the size of d0fx
    - -40 : the halo is negative !
    - -50 : the interior region has an upper index ¡ lower index
    - -80 : Can’t handle the requested order
    - -90 : periodicity is neither set to no (0) or yes (1)
    - -100 : Stencil wider than domain

13. `error`, a string which will contain the error in words. In FORTRAN, it is `character*60 error`, in C/C++, `char error[256]` suffices.

3.3 Interpolations

Within the context of simulations, interpolations are needed in two cases (1) when variables, stored at cell-corners are needed at face-centers (red dots in Fig. 1) or cell-centers (cyan squares in Fig. 1 or when one fills up a fine mesh with coarse mesh values (prolongations used in multigrid techniques and block-structured AMR). In such a case, we need to get both the red points and cyan squares from data at the blue circles (Fig. reffig:mesh). The red-circles are obtained by 1D interpolations e.g. `x_intp_cf_vc()` while the cyan squares need a 2D interpolation e.g. `xy_intp_cf_vc()`.

The case of refinement often causes confusion in terms of indexing. Let the 2-cell coarse mesh (NCELLS_C = 2) have its 3 corners (along the x-axis) indexed $0 \ldots \mathrm{NCELLS}_C$. The fine mesh has 4 cells i.e. NCELLS_F = 4 and its cell corners are indexed $0 \ldots \mathrm{NCELLS}_F$. The important thing to note here is that matters become much simpler if thought of in a cell-based manner - it is the cells which get refined and doubled, not corners.

CAUTION ! The library assumes a *factor of two* refinement, it *cannot* handle arbitrary refinement ratios.

`libVertexCenteredInterps.a` contains 3 functions :

```
FORTRAN :

c Interpolate in x
  subroutine x_intp_cf_vc( uc, d0fx, orderi, orderb,
                           biL, biR, bjL, bjR, bkL, bkR,
                           fiL, fiR, fjL, fjR, fkL, fkR,
                           iiL, iiR, ijL, ijR, ikL, ikR,
                           dfiL, dfiR, dfjL, dfjR, dfkL, dfkR,
                           iperx, ipery, iperz, ierror, message)
```

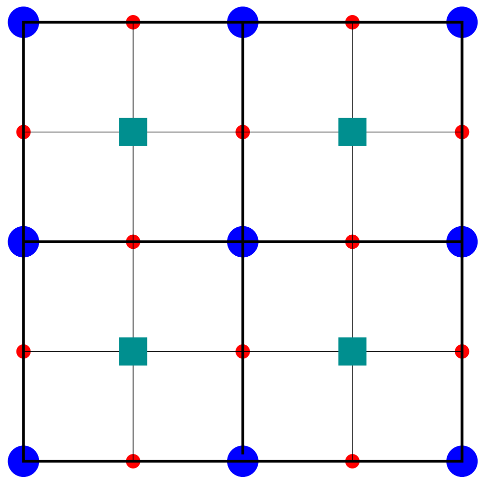

Figure 1: A vertex-centered grid on a coarse mesh (blue circles) and places that one frequently needs data interpolated to - face-centers (red circles) and cell-centers (cyan squares).

```
c Interpolate in y
  subroutine y_intp_cf_vc( -- same argument sequence as x_intp_cf_vc -- )

c Interpolate in xy
  subroutine xy_intp_cf_vc( uc, d0fx, orderi, orderb,
                            biL, biR, bjL, bjR, bkL, bkR,
                            fiL, fiR, fjL, fjR, fkL, fkR,
                            iiL, iiR, ijL, ijR, ikL, ikR,
                            dfiL, dfiR, dfjL, dfjR, dfkL, dfkR,
                            iperx, ipery, iperz, ierror, message,
                            icorner)
C/C++ :

  /* Interpolate in x */
  void  C_x_intp_cf_vc( double *uc, double *d0fx,
                        int *orderi, int *orderb,
                        int *biL, int *biR, int *bjL,
                        int *bjR, int *bkL, int *bkR,
                        int *fiL, int *fiR, int *fjL,
                        int *fjR, int *fkL, int *fkR,
                        int *iiL, int *iiR, int *ijL,
                        int *ijR, int *ikL, int *ikR,
                        int *dfiL, int *dfiR, int *dfjL,
                        int *dfjR, int *dfkL, int *dfkR,
                        int *iperx, int *ipery, int *iperz,
                        int *ierror, char *message) ;
```

```
/* Interpolate in y */
void C_y_intp_cf_vc( -- same argument sequence as C_x_intp_cf_vc -- ) ;

/* Interpolate in xy */
void C_xy_intp_cf_vc( double *uc, double *d0fx,
                      int *orderi, int *orderb,
                      int *biL, int *biR, int *bjL,
                      int *bjR, int *bkL, int *bkR,
                      int *fiL, int *fiR, int *fjL,
                      int *fjR, int *fkL, int *fkR,
                      int *iiL, int *iiR, int *ijL,
                      int *ijR, int *ikL, int *ikR,
                      int *dfiL, int *dfiR, int *dfjL,
                      int *dfjR, int *dfkL, int *dfkR,
                      int *iperx, int *ipery, int *iperz,
                      int *ierror, char *message, int *icorner) ;
```

where

1. `uc`, is the 3D `real*8` INPUT FORTRAN array containing the coarse mesh field.
2. `d0fx`, in the 3D `real*8` OUTPUT FORTRAN array containing the interpolated values.
3. `orderi, orderb`, two integers that contain the order of accuracy needed in the interior (`orderi`) and boundary (`orderb`). Currently only `orderi` is used and admits even, positive values between 2 and 10.
4. `biL, biR, bjL, bjR, bkL, bkR`, integers that denote the width of the "halo" around a rectangular domain. Each number describes the width at the lower and upper end of each axis.
5. `fiL, fiR, fjL, fjR, fkL, fkR`, integers which denote the lower and upper bounds of the indices of array `uc`
6. `iiL, iiR, ijL, ijR, ikL, ikR`, integers which denote the lower and upper bounds of the section of array `uc` where are the "interior" or "valid" points i.e. "non-halo" points.
7. `dfiL, dfiR, dfjL, dfjR, dfkL, dfkR`, integers which denote the lower and upper bounds of `uc`.
8. `iperx, ipery, iperz` integers that denote if the data is periodic in x-, y- or z-directions. setting them to 1 means "Yes", 0 is "No". Anything else is an error.
9. `ierror`, an error return code. Zero means "OK", else
   - -10 : the interior + halo exceed the size of the array
   - -20 : the interior is wider than the size of d0fx
   - -40 : the halo is negative !
   - -50 : the interior region has an upper index ¡ lower index
   - -80 : Can't handle the requested order

- -90 : periodicity is neither set to no (0) or yes (1)
- -100 : Stencil wider than domain

10. `message`, the error message in words. For FORTRAN, `character*60 message`, for C/C++, `char message[256]` suffices.

11. `icorner`, an integer, when set to 1, indicates that the corner value is also needed. Else set to zero.

References

[1] Libraries of high-order discretizations, interpolations and filters. http://www.caip.rutgers.edu/~jaray/HighOrder/SISC06.html.

[2] J. Ray, C. A. Kennedy, S. Lefantzi, and H. N. Najm. Using high-order methods on adaptively refined block-structured meshes–discretizations, interpolations, and filters. *SIAM Journal on Scientific Computing*, 2006. in review, also available as a Sandia Technical Report, SAND2005-7981.