GSLIB Findpoints in MFEM

1 Introduction

GSLIB findpts has been implemented in MFEM using a C++ class (*findpts_gslib*) that calls the C code. A key aspect of modifying inputs from MFEM to be compatible with GSLIB is the way mesh and variables defined on the mesh are stored. While we use gridfunctions in MFEM, GSLIB uses the actual locations (x, y, z) of the GLL points for each element in the mesh. This means that gridfunctions must be converted to the actual nodal values before calling the *gslib* C code. Additionally while we only need to store the unique nodes in MFEM, GSLIB requires nodes for each element in a lexicographic ordering, and shared nodes between elements are counted more than once. The length of any scalar defined on the mesh with E elements is (EK^d) where K is the number of GLL points in each element in each direction, and d is the space dimension. To read about inner machinery of *gslib* see [1].

2 Installation

The *gslib* tar must be untarred inside the MFEM build directory. There is a file that has been modified in the source *gslib - src/types.h* that looks for the *config/_config.hpp* to determine wether the MFEM installation is serial or parallel, and accordingly configures *gslib*. Thus, it is important that *gslib* is either untarred in MFEM build directory, or the MFEM source is modified accordingly.
gslib can then be installed as:

```
make CC=mpicc
or
make CC=gcc
```

for parallel or serial installation respectively. This should be followed by:

```
make install
```

Note: Every time MFEM build is changed from serial to parallel, or vice-versa, *gslib* must be rebuilt accordingly as well.

3 Changes to makefile

The makefile and defaults.mk file have been changed to link gslib with MFEM.

4 Setting up findpts

gslib_findpts can be used in MFEM by creating a new object as:

```
gsfl = new findpts_gslib(MPI_COMM_WORLD); in parallel
or
gsfl = new findpts_gslib(); in serial
```

Next *findpts* can be setup as:

```
gsfl->gslib_findpts_setup(pfespace,pmesh,qorder,bb_t,newt_tol,mpt_max);
```

- *pfespace* is the finite element space object
- *pmesh* is the mesh object
- *qorder* is the number of quadrature points to be used in each direction for each element, for *GSLIB findts* library.
- *bb_t* - relative size by which to expand bounding boxes that are setup around each element. e.g. $bb\_t = 0.05$
- *newt_tol* - Tolerance for newton solver. e.g. $newt\_tol = 1.e-12$
- *npt_max* - number of points to iterate on simultaneously... impacts the memory allocation. e.g. $npt\_max = 256$

This constructor converts the mesh from gridfunction to a vector of doubles using MFEM's routine *GetValue* and saves it. Additionally this also calls a *GSLIB* routine that setup its machinery required later on:
Alternatively the user can say:

```
gsfl->gslib_findpts_setup(pfespace,pmesh,qorder);
```

This sets up *findpts* with default choices of $bb\_t = 0.05$, $newt\_tol = 1.e-12$, and $npt\_max = 256$.

5 Using findpts to query which element contains a given $\vec{x}$ location

In order to query and determine which elements on which processor hold a list of given $\vec{x}$ location, the user must set up a few arrays/vectors which will hold the output. Given n points to be found, setup:

- `Vector xyzlist(n*dim)`

  Holds the x, y, z coordinates of all the points to be found.. The ordering should be $x_{i=1...n}$ $y_{i=1...n}$ $z_{i=1...n}$

- `Array<uint> pel(n)`

  Will hold the element number that the point was found in

- `Array<uint> pproc(n)`

  Will hold the processor number that the point was found on

- `Vector pr(n*dim)`

  Holds the r, s, t coordinates (reference space) inside the element for the point. The ordering inside this vector is $r_{i=1}, s_{i=1}, t_{i=1}, r_{i=2}, s_{i=2}, t_{i=2}, \ldots\ldots r_{i=n}, s_{i=n}, t_{i=n}$

- `Array<uint> pcode(n)`

  Tells wether the point was found inside an element (0), on the border of an element(1), or not found (2).

- `Vector pdist2(n)`

  The distance squared from found to sought point (in xyz space)

Find the points by:

```
gsfl->gslib_findpts(&pcode,&pproc,&pel,&pr,&pdist2,&xyzlist,nxyz);
```

Note: The reference space coordinates returned by *gslib* are in $[-1, 1]$. This must be kept in mind if these coordinates are used in MFEM, because the default range in MFEM is $[0, 1]$.

6 Interpolating a scalar field using findpts_eval

This method uses the output from *gslib_findpts* to interpolate any given gridfunction at the locations specified in *Vector xyzlist*. This requires the user to setup a Vector that can hold the interpolated values:

```
Vector int_field(n)
```

The scalar gridfunction (*gf_input*) can then be interpolated as:

```
gsfl->gslib_findpts_eval(&int_field,&pcode,&pproc,&pel,&pr,&gf_input,nxyz);
```

This method populates the Vector *int_field* with interpolated values at the queried locations.

Note If the user inputs a gridfunction to method *gslib_findpts_eval*, the method converts the gridfunction to actual nodal values at each grid point. If the user must interpolate the same gridfunction more than once, it will be effective to convert the gridfunction to nodal values and save them in a vector field.

```
Vector vec_input(gsfl->GetFptMeshSize());
gsfl->gf2vec(gf_input,&vec_input);
```

This only needs to happen once. The vector field can then be input to *gslib_findpts_eval* as:

```
gsfl->gslib_findpts_eval(&ind_fpt,&pcode,&pproc,&pel,&pr,&vec_input,nxyz);
```

The output *ind_fpt* can be converted to gridfunction using MFEM methods such as *Distribute*.

7 Clearing up memory

Memory used by *gslib findpts* can be freed by:

```
gsfl->gslib_findpts_free ()
```

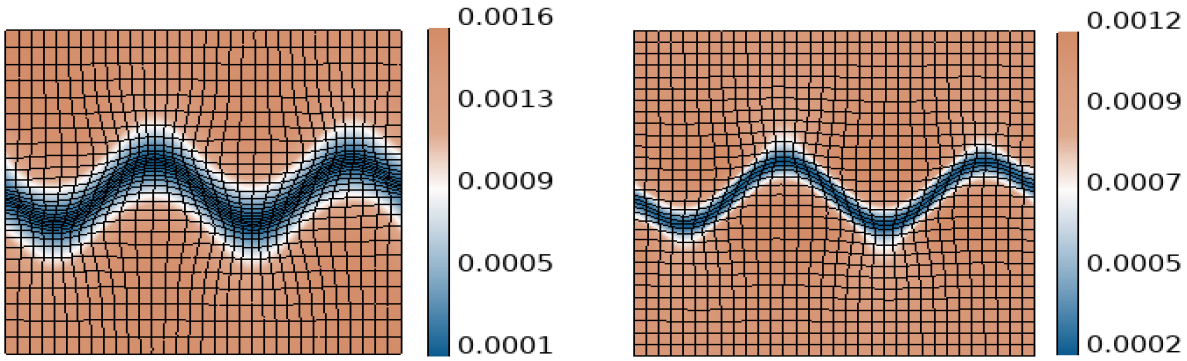
8 Example

Given an original mesh gridfunction *x0* and a field on it *fld0*, interpolate the scalar field onto the modified mesh vector *xnew*.

```
// Declarations
int nxyz = xnew.Size()/dim;
Array<uint> pel(nxyz);
Array<uint> pcode(nxyz);
Array<uint> pproc(nxyz);
Vector pr(nxyz*dim);
Vector pd(nxyz);

// Constructor
gsfl = new findpts_gslib(MPI_COMM_WORLD);
// Set up findpts
gsfl->gslib_findpts_setup(x0->ParFESpace(),x0->ParFESpace()->GetParMesh(),8);
// Findpts
gsfl->gslib_findpts(&pcode,&pproc,&pel,&pr,&pd,&xnew,nxyz);
// Interpolate
Vector ind_fpt(nxyz);
gsfl->gslib_findpts_eval(&ind_fpt,&pcode,&pproc,&pel,&pr,&ind0,nxyz);
// Convert interpolated vector to gridfunction
ind->Distribute(ind_fpt);
```

I have also been able to use *findpts* in the mesh optimizer to replace the advection step where we remap the indicator function. Here I show improvement I see for one of the test cases with *findpts*:



(a) With *findpts* (b) Without *findpts*

Figure 1: Indicator function on smoothed mesh

References

[1] Azad Noorani, Adam Peplinski, and Philipp Schlatter. Informal introduction to program structure of spectral interpolation in nek5000. 2015.