Classification: Open

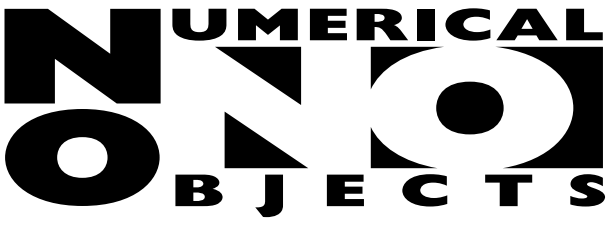
The Numerical Objects Report Series

Report 2000-02

A Supplementary Note for the Diffpack Adaptivity Toolbox

Xing Cai

February 16, 2001



Numerical Objects Report 2000-02

Title

A Supplementary Note for the Diffpack Adaptivity Toolbox

Written by

Xing Cai

Approved by

Are Magnus Bruaset
2000.12.13

Contact information

Phone: +47 22 06 73 00
Fax: +47 22 06 73 50
Email: info@nobjects.com
Web: http://www.nobjects.com

Contents

This report should be referenced as shown in the following BibTeX entry:

```
@techreport{NO2000-02,
  author = "Xing Cai",
  title = "A Supplementary Note for the Diffpack Adaptivity Toolbox",
  institution = "Numerical Objects AS, Oslo, Norway",
  type = "The Numerical Objects Report Series",
  number = "\#{}2000-02",
  year = "February 16, 2001",
}
```

A Supplementary Note for the Diffpack Adaptivity Toolbox

Xing Cai

This short note contains supplementary information about how to use the Diffpack Adaptivity Toolbox. Readers are assumed to have already read Chapter 3.6 in [1].

Chapter 1

A simple example

We consider a simple example where a 2D/3D elliptic boundary value problem is solved repeatedly for a given number of times. Between two consecutive solves, the grid is refined adaptively according to a chosen strategy. It is assumed that the starting finite element (FE) grid contains either triangular or tetrahedral elements. Therefore, `GridFEAdT` is chosen as the concrete adaptive grid type. It should also be noted that the other adaptive grid type `GridFEAdB` can readily substitute `GridFEAdT` if necessary.

1.1 Extensions in the header file

Denote our simulator class capable of adaptive mesh refinement by `AdapEx`. The header file `AdapEx.h` can be as follows.

```
#include <FEM.h>
#include <LinEqAdmFE.h>
#include <GridFEAdT.h>
#include <GridRefAdm.h>

class AdapEx : public FEM
{
protected:
  int no_of_solves;
  Handle(GridFE)     grid;
  Handle(FieldFE)    u;
  Handle(DegFreeFE)  dof;
  Handle(LinEqAdmFE) lineq;
  Vec(real)          linsol;
  Handle(GridRefAdm) refcrit1;
```

```
  void redim ();
  virtual void fillEssBC ();
  virtual void integrands (ElmMatVec& elmat, const FiniteElement& fe);

public:
  AdapEx () {}
 ~AdapEx ();
  virtual void define (MenuSystem& menu, int level = MAIN);
  virtual void scan   ();
  virtual void adm    (MenuSystem& menu);
  virtual void solveProblem ();
  virtual void resultReport ();
  virtual void evalOwnRefInd (FieldPiWisConst& refinement_field,
      real& evaluated_error);
};
```

Compared with a standard Diffpack simulator class for solving an elliptic boundary value problem, `AdapEx` has the following additions caused by the adaptive grid refinement.

1. Inclusion of two new header files:

   ```
   #include <GridFEAdT.h>
   #include <GridRefAdm.h>
   ```

2. Addition of a handle to a so-called grid refinement administrator:

   ```
   Handle(GridRefAdm) refcrit1;
   ```

3. Definition of a new virtual member function:

   ```
   virtual void evalOwnRefInd (FieldPiWisConst& refinement_field,
                               real& evaluated_error);
   ```

1.2 Extensions in the source code

We first note that member function `evalOwnRefInd` defines a user's own grid refinement indication procedure. It can therefore be *omitted* if the user only applies the existing refinement indicators provided by the adaptivity toolbox. As a concrete example of this virtual function, we suppose that the user wants

to use the gradient of the solution as the basis for indicating elements to be refined, but he at the same time wants to avoid refining elements whose size is already below a threshold, say 10^{-6}.

```
void AdapEx:: evalOwnRefInd (FieldPiWisConst& refinement_field,
     real& evaluated_error)
{
  // use gradient of the solution
  FEM::makeGradient (refinement_field, *u);
  Vec(real) elm_volume (refinement_field.values().size());
  ErrorEstimator::calcElmVolume (elm_volume, *grid);
  evaluated_error = 0.;
  for (int e=1; e<=refinement_field.values().size(); e++)
    if (elm_volume(e)<1.0e-6)
      // no refinement for small elements
      refinement_field.setValueElm (e,0.);
    else
      evaluated_error += sqr(refinement_field.values()(e));
  evaluated_error = sqrt(evaluated_error);
}
```

Here we mention that the piecewise constant FE field `refinement_field` is to contain refinement indicator information such that each element is assigned with a non-negative value representing the size of the error. The larger the indicator value, the more likely that an element is to be refined. So giving those elements whose size (found by the `ErrorEstimator::calcElmVolume` function) is below the given threshold will prevent them from being refined further.

Assume that integer `no_of_solves` contains the number of solves that the user wants to solve the same problem on a series of consecutively refined FE grids. Then the main function of `AdapEx` can be as follows.

```
void AdapEx:: solveProblem ()
{
  for (int i=1; i<=no_of_solves; i++) {
    fillEssBC ();
    makeSystem (*dof, *lineq);
    linsol.fill (0.);
    lineq->solve ();
    dof->vec2field (linsol, *u);
    refcrit1->refine (grid);
    redim ();
```

```
  }
}
```

It should be noted that the function `redim` is of great importance and ensures that dimensions of the internal data are adjusted accordingly after the grid is adaptively refined. In the present case, we may have a following definition of `redim`:

```
void AdapEx:: redim ()
{
  if (!dof.ok())
    dof.rebind (new DegFreeFE (*grid, 1));
  else
    dof->redim (*grid, 1);
  if (!u.ok())
    u.rebind (new FieldFE (*grid, "u"));
  else
    u->redim (*grid, "u");
  linsol.redim (dof->getTotalNoDof());
  lineq->attach (linsol);
}
```

In addition to the above major extensions in the source code of `AdapEx`, here are two necessary minor extensions:

1. Inside the `AdapEx::define` function:

   ```
   GridRefAdm:: defineStatic (menu, level+1);
   ```

2. Inside the `AdapEx::scan` function:

   ```
   grid.rebind (new GridFEAdT());
   refcrit1.rebind (new GridRefAdm());
   refcrit1->scan (menu);
   redim ();
   refcrit1->getRefinementInd().attach (*u);
   refcrit1->getRefinementInd().attach (*this);
   ```

Chapter 2

Control of adaptive grid refinement through menu input

The extensive list of menu items associated with class `GridRefAdm` provides users with full control of how adaptive grid refinement can be carried out. The result of invoking `GridPartAdm::defineStatic` is that two submenus are created for choosing the refinement indicator and adjusting the refinement parameters, respectively.

2.1 The `RefinementInd_prm` submenu

The following items are inside the `RefinementInd_prm` submenu:

`refinement indicator` determines which refinement indicator will be used to mark the elements with indicator values. Possible choices are listed in Chapter 3.6.2[1] of [1].

`hypercube regions` is only useful when `refinement indicator` is chosen as `GeometricRegions`. See Chapter 3.6.2 of [1]. See also comments in the header file `RefinementInd.h`.

`disk regions` is only useful when `refinement indicator` is chosen as `GeometricRegions`. See Chapter 3.6.2 of [1].

`contour value` is only useful when `refinement indicator` is chosen as `Contour`. For this type of indicator, elements that are in the vicinity of a contour line/surface of an attached field are marked for refinement.

[1] Note that there is a typo at the top of page 291 of [1].

`error used in stop criterion` is only useful when the grid to be refined is of type `GridFEAdT`. It determines whether the actual size of the estimated error, e.g. `evaluated_error` returned from `evalOwnRefInd`, is used to stop the iterative refinement procedure. The default value is `OFF`.

2.2 The `GridRefAdm` submenu

The content of this submenu can be divided into three parts.

2.2.1 Menu items that are used by both `GridFEAdT` and `GridFEAdB`

`refinement strategy` determines whether refinement is based on the absolute indicator value of each element. Choice `Absolute` means yes, whereas choice `Percent` means that the indicator values are only considered relatively against each other.

`hi_limit` is only used when `refinement strategy` is chosen to be `Absolute`. See below.

`lo_limit` is only used when `refinement strategy` is chosen to be `Absolute`. All elements that have indicator values lying between `lo_limit` and `hi_limit` are to be refined.

`percent refined elements` is only used when `refinement strategy` is chosen to be `Percent`. All the indicator values are sorted in an increasing order. The elements that have their indicator values among the top percentage given by `percent refined elements` are to be refined.

2.2.2 Menu items that are only used by `GridFEAdT`

`tolerance` gives a tolerance value for stopping the iterative refinement procedure. It is only used when the `error used in stop criterion` item is chosen to be `ON`.

`max refinement levels` gives the maximum number of iterations allowed in the iterative refinement procedure.

`max nodes` will stop the iterative refinement procedure if the number of grid nodes is larger than a given threshold.

`max elements` will stop the iterative refinement procedure if the number of elements is larger than a given threshold.

`mixed refinement method` determines whether elements that are chosen for refinement may produce a variable number of child elements. The default choice is `ON`.

`refine all elements` determines whether all the elements are to be refinement, neglecting the calculated indicator values.

`regular or bisection` is only useful when `refine all elements` is chosen as `ON`. It has three choices: `REGULAR`, `BISECTION` and `SHORTEST`.

`no of edge refinements` is only useful when `mixed refinement method` is chosen as `OFF` or if `refinement strategy` is chosen as `Absolute`. This results in that all the elements that are to be refined will produce a fixed number of child elements.

2.2.3 Menu items that are only used by `GridFEAdB`

`subgrid partition` determines how many child elements that will arise from each marked element.

`number of refinements` gives the number of iterations within one adaptive grid refinement procedure.

`start grid for refinement` accepts two choices: `current` and `original`. It determines whether refinement is carried out on the current level of the grid hierarchy or on the coarsest grid level. The default value is `original`.

`smooth grid` determines whether the intermediate result during the iterative grid refinement procedure needs smoothing to improve the grid quality.

`max smooth number` gives a maximum number of smoothings allowed.

Chapter 3

Questions and answers

1. *I use* `set subgrid partition=2` *when trying to refine a* `GridFEAdB` *grid with* `ElmB9n2D` *elements, but no element is refined, why?*
   Use `set subgrid partition=4` instead. This is because each `ElmB9n2D` element has already used 2 subdivisions in each direction.

2. *I want to apply* `GridRefAdm::refine` *repeatedly upon a* `GridFEAdB` *grid, why does my program always abort abnormally?*
   Remember to use `set start grid for refinement = current`.

3. *Is it possible to refine a* `GridFEAdT` *grid with* `ElmT10n3D` *elements?*
   The current version of Diffpack adaptivity toolbox does not support direct refinement of a `GridFEAdT` grid with `ElmT10n3D` elements. As a remedy, a user should refine a `GridFEAdT` grid with `ElmT4n3D` elements and use the `changeGridL2Q` function of class `GridFEAdT` to generate a new grid with `ElmT10n3D`.

Bibliography

[1] H. P. Langtangen, *Computational Partial Differential Equations – Numerical Methods and Diffpack Programming*. Springer (1999).