Developers' Report Manual - CEMRACS 2007 a posteriori estimator

Radek Fučík, Ibrahim Cheddadi

1 List of Files

The a posteriori estimates are coded in FreeFEM++ using dynamic loading (see FreeFEM++ manual how to compile the code).

1.1 Source code

- `Makefile`
  Scriptfile containing information how to build cemracs module (invoked by `make` command).
- `cemracs2007.cpp`
  CEMRACS 2007 a posteriori estimate class.
- `cemracs2007cl.cpp`
  Implementation of the classical error estimate.
- `cemracs2007aux.cpp`
  Auxiliary functions needed for both cemracs2007.cpp and cemracs2007cl.cpp files.

1.2 Script files

- `L.edp`
  L-Shape test problem - pure diffusion case (run by `FreeFEM++` L.edp)
- `SQ.edp`
  Square test problem - reaction-diffusion case (run by `FreeFEM++` SQ.edp)

2 FreeFEM++ functions

2.1 cemracs2007

- ApposterioCemracs2007() function Syntax:

```
load "cemracs2007";
AposterioCemracs2007(
        Th, [u, dx(u), dy(u), rhs, u0 u0dx, u0dy, react],
        filename="name of output file",
        output=[ etak[], etav[], etaDF[], etaR[], errk[], errv[],
                 hd[], etaDF1[], etaDF2[] ],
        medit=MeditExport, strategy=Strategy);
```

  Description:

  - Th [`mesh`] . . . FreeFEM++ mesh
  - u [`fespace(Th,P1)`] . . . numerical solution on Th

- dx(u) ... partial derivative $\partial u/\partial x$
- dy(u) ... partial derivative $\partial u/\partial y$
- rhs [`func`] right-hand side of the equation $-\Delta u + ru = f$ (i.e., the source term f)
- u0 [`func`] ... analytical solution
- u0dx [`func`] ... partial derivative of analytical solution $\partial u0/\partial x$
- u0dy [`func`] ... partial derivative of analytical solution $\partial u0/\partial y$
- react [`func`] ... reaction term r in $-\Delta u + ru = f$
- filename [`string`] ... name of the output file
- output [`array`] ... array of output vectors
  - etak[] [`fespace(Th,P0)`] ... estimated error per triangles (for visualization purposes)
  - etav[] [`fespace(Th,P1)`] ... estimated error per vertices
  - etaDF[] [`fespace(Th,P1)`] ... estimator η_{DF}
  - etaR[] [`fespace(Th,P1)`] ... estimator η_R
  - errk[] [`fespace(Th,P0)`] ... exact energy error per triangles computed from the exact solution `(u0,u0dx,u0dy)` and numerical solution `u,dx(u),dy(u)`
  - errv[] [`fespace(Th,P1)`] ... exact energy error per vertices computed from the exact solution `(u0,u0dx,u0dy)` and numerical solution `u,dx(u),dy(u)`
  - hd[] [`fespace(Th,P1)`] ... characteristic length of mesh Th distribution (per vertex) - used for adaptation of mesh
  - etaDF1[] [`fespace(Th,P1)`] ... estimator $\eta_{DF}^{(1)}$
  - etaDF2[] [`fespace(Th,P1)`] ... estimator $\eta_{DF}^{(2)}$
- medit [`int`] ... indicates if medit output files are created [1=yes, 0=no]
- strategy [`int`] ... indicates which estimator t_h computation strategy is used (see cemracs2007.cpp):
  - 0 : "jump" strategy
  - 1 : minimization of ${\eta_{DF}^{(1)}}^2 + \eta_R^2$ strategy
  - 2 : minimum of "jump" and minimization of ${\eta_{DF}^{(1)}}^2 + \eta_R^2$ strategy
  - 3 : minimization of $\eta_{DF}^2 + \eta_R^2$ strategy
  - 4 : minimum of all strategies (0,1,2,3 and 5)
  - 5 : prescription of t_h such that η_R vanishes

3 Brief description of internal functions (source code)

3.1 cemracs2007.cpp

-
  ```
  double rhs_reaction(double x, double y, MeshPoint* mp,
                      vector<Expression> sol, Stack &stack)
  ```
  Returns the value of $f - rp_h$ (given as argument in the FreeFEM++ script) at the point of coordinates (x, y).

-
  ```
  double reaction(double x, double y, MeshPoint* mp,
                  vector<Expression> sol, Stack &stack)
  ```
  Returns the value of the reaction function r (given as argument in the FreeFEM++ script) at the point of coordinates (x, y).

-
  ```
  double ex(double x, double y, MeshPoint* mp,
            vector<Expression> sol, Stack &stack)
  ```
  Returns the value of the exact solution (given as argument in the FreeFEM++ script) at the point of coordinates (x, y).

-
  ```
  double ex_dx(double x, double y, MeshPoint* mp,
               vector<Expression> sol, Stack &stack)
  ```
  Returns the value of the first partial derivative after x of the exact solution (given as argument in the FreeFEM++ script). at the point of coordinates (x, y).

-
  ```
  double ex_dy(double x, double y, MeshPoint* mp,
               vector<Expression> sol, Stack &stack)
  ```
  Returns the value of the first partial derivative after y of the exact solution (given as argument in the FreeFEM++ script) at the point of coordinates (x, y).

-
  ```
  double ph_dx(double x, double y, MeshPoint* mp,
               vector<Expression> sol, Stack &stack)
  ```
  Returns the value of the first partial derivative after x of the numerical solution (given as argument in the FreeFEM++ script) at the point of coordinates (x, y).

-
  ```
  double ph_dy(double x, double y, MeshPoint* mp,
               vector<Expression> sol, Stack &stack)
  ```
  Returns the value of the first partial derivative after y of the numerical solution (given as argument in the FreeFEM++ script) at the point of coordinates (x, y).

-
  ```
  double compute_m_D_square(double C_F_or_C_P, double h_D, double c_r)
  ```
  Returns the constant m_D^2.

-
  ```
  double compute_trace_constant(double edge_length, double h_K, double area)
  ```
  Returns the trace constant C_t.

-
  ```
  double compute_m_K(double C_F_or_C_P, double h_K, double c_r)
  ```
  Returns the constant m_K.

-
  ```
  double compute_m_K_tilde(double h_K, double c_r)
  ```
  Returns the constant $\tilde{m}_K$.

-
  ```
  double qformula_exact_uni(double x1, double y1, double x2, double y2,
                            double x3, double y3, MeshPoint* mp,
                            vector<Expression> sol, Stack &stack )
  ```
  Returns the value of $\int_K \left((\partial_x p_h - \partial_x p)^2 + (\partial_y p_h - \partial_y p)^2\right) \mathrm{dx}$ on a triangle K given by the coordinates (x_1, y_1),... of its vertices, where p is the exact solution. The integral is evaluated by a quadrature formula.

-
  ```
  double qformula_DF(double x1, double y1, double x2, double y2,
                     double x3, double y3, double alpha, double beta,
                     double gamma, double gradphx, double gradphy,
                     double aireK)
  ```

Returns the value of $\int_K \left((\partial_x p_h + \mathbf{t}_h^x)^2 + (\partial_y p_h + \mathbf{t}_h^y)^2\right) \mathrm{d}\mathbf{x}$ on a triangle K given by the coordinates (x_1, y_1),... of its vertices. The integral is evaluated by a quadrature formula.

-
  ```
  double qformula_R_uni_react(double x1, double y1, double x2, double y2,
                              double x3, double y3, double alpha,
                              double beta, double gamma, double aireK,
                              MeshPoint* mp, vector<Expression> sol,
                              Stack &stack)
  ```

  Returns the value of $\int_K (f - rp_h - \nabla \cdot \mathbf{t}_h)^2 \,\mathrm{d}\mathbf{x}$ on a triangle K given by the coordinates (x_1, y_1),... of its vertices. The integral is evaluated by a quadrature formula.

-
  ```
  void tridiag_solver(int N, double *a, double *b, double *c,
                      double *f, double *y)
  ```

  Computes in `double *y` the solution of a tridiagonal system where ... *(Radek you may know)*

-
  ```
  int cemracs_solver(int nu, double *alpha, double *beta,
                     double *gamma, double *phi, double *upsilon)
  ```

  (Radek you may know)

-
  ```
  void compute_matrix_int_tri(double *a,double *b,double *c,
                              double *f, double **E_coef, int N)
  ```

  Assembles the tridiagonal matrix (in `double *a,double *b,double *c`: as in `tridiag_solver`) and the vector (`double *f`) corresponding to the quadratic form that is to be minimized in the case of an interior dual volume.

-
  ```
  void compute_matrix_ext_tri(double *a,double *b,double *c,
                              double *f, double **E_coef, int N,
                              double E10t, double E4n2t, double E1n2t)
  ```

  Assembles the tridiagonal matrix (in `double *a,double *b,double *c`: as in `tridiag_solver`) and the vector (`double *f`) corresponding to the quadratic form that is to be minimized in the case of an exterior dual volume.

-
  ```
  void compute_X0(double *X0, double vx, double vy, int N,
                  double **A_triangle_coor, MeshPoint* mp,
                  vector<Expression> sol, Stack &stack, int boundary)
  ```

  Computes the vector `double *X0` of the prescribed coefficients α_0 for each subtriangle. `X0[i]` contains α_0^i *(is it right ?).*

-
  ```
  void compute_coef_1(int i, double **Res_coef,
                      double **Diff_coef, double **E_coef,
                      double **alpha_tab, MeshPoint* mp,
                      vector<Expression> sol, Stack &stack,
                      double xx0, double yy0, double xx1, double yy1,
                      double xx2, double yy2, double gx, double gy,
                      double mD2)
  ```

  Computes for the `i`-th subtriangle (given by the coordinates (xx0,yy0),... of its vertices) the corresponding part of the coefficients array `double **E_coef` that is used for assembling the matrix and the linear part of the quadratic form $\eta_R^2 + \left(\eta_{DF}^{(1)}\right)^2$.

-
  ```
  void compute_coef_2(int i, double **Res_coef, double **Diff_coef,
                      double **E_coef, double **alpha_tab,
                      MeshPoint* mp, vector<Expression> sol, Stack &stack,
                      double xx0, double yy0, double xx1, double yy1,
                      double xx2, double yy2, double gx, double gy,
                      double mD2, double C_F_or_C_P, double c_r)
  ```

  Computes for the `i`-th subtriangle (given by the coordinates (xx0,yy0),... of its vertices) the corresponding part of the coefficients array `double **E_coef` that is used for assembling the matrix and the linear part of the quadratic form $\eta_R^2 + \left(\eta_{DF}^{(2)}\right)^2$.

-
  ```
  void compute_grad_ph(int i, double &gx, double &gy, double vx,
                       double vy, double **A_triangle_coor,
                       MeshPoint* mp, vector<Expression> sol, Stack &stack)
  ```

  Computes in `double &gx, double &gy` the constant value of ∇p_h on the `i`-th triangle.

-
  ```
  double compute_estimators(double** alpha_tab, double C_F_or_C_P,
         double **A_triangle_coor, double vx, double vy,
         int n_big_tria,MeshPoint* mp,
         vector<Expression> sol, Stack &stack, int boundary,
         double &est_DF,// square root of the DF estimator
         double &est_R,// square root of the R estimator
         double &est_DF_1,// square root of the DF_1 estimator
         double &est_DF_2 // square root of the DF_2 estimator
  )
  ```

  Computes the values of the estimators from $\mathbf{t}_h$, given the array of its coefficients `double** alpha_tab`.

-
  ```
  void compute_alpha_tab_from_X(double **alpha_tab,double alpha1,
                                double alpha2, double *X,
                                int n_big_tria, int boundary)
  ```

  Computes `double **alpha_tab`, where `alpha_tab[i][j]` contains α_j^i, from the solution `double *X` of the minimization problem.

-
  ```
  void compute_alpha0(double **alpha_tab, int n_big_tria,
       double **A_triangle_coor,  double vx, double vy,
       MeshPoint* mp, vector<Expression> sol, Stack &stack)
  ```

  Adds in `double **alpha_tab` the values of α_0^i.

-
  ```
  void compute_th_minimization_1(double **alpha_tab, int n_big_tria,
       double **A_triangle_coor, double vx, double vy,
       MeshPoint* mp, vector<Expression> sol, Stack &stack,
       int boundary, double C_F_or_C_P
       )
  ```

  Performs the minimization process of $\eta_R^2 + \left(\eta_{DF}^{(1)}\right)^2$. The output is the array `double** alpha_tab` of the coefficients of $\mathbf{t}_h$.

-
  ```
  void compute_th_minimization_2(double **alpha_tab, int n_big_tria,
       double **A_triangle_coor, double vx, double vy,
       MeshPoint* mp, vector<Expression> sol, Stack &stack,
  ```

```
int boundary, double C_F_or_C_P
)
```

Performs the minimization process of $\eta_R^2+\left(\eta_{DF}^{(2)}\right)^2$. The output is the array `double** alpha_tab` of the coefficients of $\mathbf{t}_h$.

-
  ```
  void compute_th_jumping(double **alpha_tab, int n_big_tria,
       double **A_triangle_coor,  double vx, double vy,
       MeshPoint* mp, vector<Expression> sol, Stack &stack,
       int boundary, double C_F_or_C_P)
  ```

  Computes the $\mathbf{t}_h$ with the "jump" strategy (no minimization process).

3.2 cemracs2007aux.cpp

-
  ```
  void intersection(double a1, double a2, double b1, double b2,
       double c1, double c2, double d1, double d2, double &x1, double &x2)
  ```

  Returns coordinates of an intersection $[x_1, x_2]$ of two lines that are given by points $[a_1, a_2]$, $[b_1, b_2]$ and the other by $[c_1, c_2]$, $[d_1, d_2]$.

- `double distance(double a1, double a2, double b1, double b2)`

  Returns the distance between two points.

-
  ```
   void barycenter(double &x1, double &x2, double a1, double a2,
        double b1, double b2, double c1, double c2)
  ```

  Computes the barycenter $[x_1, x_2]$ of a triangle.

-
  ```
   double angle(double a1, double a2, double b1, double b2,
         double c1, double c2)
  ```

  Computes the angle between vectors $[c_1, c_2]$, $[a_1, a_2]$ and $[c_1, c_2]$, $[b_1, b2]$.

- `double vectorangle(double u1, double u2)`

  Returns the argument of a vector (u_1, u_2).

-
  ```
  int compute_angle_interval(double &angle_d, double &angle_u,
      double c1, double c2, double a1, double a2, double b1, double b2)
  ```

  Returns the interval of internal angles corresponding to the vertex $[c_1, c_2]$.

- `int isin(double u, double a1, double a2)`

  Inidicates, if an angle u is between a_1 and a_2.

-
  ```
  int interval_intersection(double &x1,double &x2, double a1, double a2,
      double b1,double b2)
  ```

  Intersects two angle intervals together.

-
  ```
  int is_intersection_nonempty(int ne, int n,int k,double **T_d,
      double **T_u,int **T_is,double lower, double upper)
  ```

  Indicates if the intersection of a set of angle intervals is non-empty.

- `int iscondition27(int n, double **A_triangle_coor, double v1, double v2, int ne, double **A_edges)`

  Indicates, if the condition !!!!!!!!!!!!!

- `int isconvex(int nt, double **A_triangle_coor, double v1, double v2, int boundary)`

  Indicates if the dual volume is convex.

- `double compute_C_P(int n, double **A_triangle_coor, double v1, double v2,int boundary)`

  Computes the Poincare constant.

- `double compute_diameter(int n, double **A, double v1, double v2)`

  Computes diameter of the dual volume.

- `double compute_minimal_diameter(int n, double **A, double v1, double v2)`

  Determines the minimal diameter of dual volumes.

- `double area_K(double x1, double y1, double x2, double y2, double x3, double y3)`

  Computes the area of a triangle.

- `double compute_max_edge_triangle(double x1, double y1, double x2, double y2, double x3, double y3)`

  Determine the longest edge of a triangle.

- `double compute_min_edge_triangle(double x1, double y1, double x2, double y2, double x3, double y3)`

  Determine the shortest edge of a triangle.

- `double compute_C_F(int n, double **A, double v1, double v2,int boundary)`

  Compute the Friedrichs constant.

4 Example FreeFEM++ scripts

4.1 diff.edp

Classical L shape domain problem, $\Delta u = 0$.

```
// load cemracs -module
load "cemracs2007";

// the classical  L space problem
border a(t=-1,1){x=t; y=-1; label=1;};
border b(t=-1,0){x=1; y=t; label=2;};
border c(t=0,1){x=1-t; y=0; label=3;};
border d(t=0,1){x=0; y=t; label=4;};
border e(t=0,1){x=-t; y=1; label=5;};
border f(t=-1,1){x=-1; y=-t; label=6;};
mesh Th=buildmesh(a(4)+ b(2) + c(2) + d(2) + e(2) + f(4));

// FE SPACES DEFINITION
fespace Vh(Th,P1); // the solution
fespace Pvh(Th,P1); // the indicator in each vertex
fespace Pth(Th,P0); // the indicator in each triangle

//  FE approximations
Vh u,v;

// EXACT SOLUTION DEFINITION
func real tita(real t) // atan2 continous in y=0 :
{
real q=0;
if (t<=0) q=t+2*pi;
if (t>0) q=t;
return q;
};
// exact solution function
func u0=sin((2.0/3.0)*tita(atan2(y,x)))*(x^2+y^2)^(1.0/3.0);
// exact solution derivative du0/dx
func u0dx=
(2.0/3.0)*((x*x + y*y)^(-2.0/3.0))*(x*sin(2.0/3.0*tita(atan2(y,x)))-
y*cos(2.0*tita(atan2(y,x))/3.0));
// exact solution derivative du0/dy
func u0dy=
(2.0/3.0)*((x*x + y*y)^(-2.0/3.0))*(y*sin(2.0*tita(atan2(y,x))/3.0)+
x*cos(2.0*tita(atan2(y,x))/3.0));
// reaction term (zero in this case)
func react=0;
// right hand side of the laplace equation (source term)
func rhs= 0;

// FE PROBLEM
problem Laplace(u,v)=
    int2d(Th)(u*v*react + dx(u)*dx(v) + dy(u)*dy(v))-int2d(Th)(rhs*v)
    +on(1,2,3,4,5,6,u=u0);

for (int i=0;i<10;i++) // refining the mesh
{
 u=u;
 Laplace;

 // ESTIMATOR VARIABLES
 Pvh etav;   // squares of final estimator (per vertex)
 Pvh etaDF;  // squares of DF part of the estimator
 Pvh etaR;   // squares of R part of the estimator
 Pvh errv;   // squares of computed error (per vertex)
 Pvh hd;     // computed mesh size
 Pvh etaDF1; // squares of DF(1) part of the est.
 Pvh etaDF2; // squares of DF(2) part of the est.
 Pth etak;   // squares of final estimator (per triangle)
 Pth errk;   // squares of computed error (per triangle)


 int estimator = 0;
 // 0 ... jump estimator
 // 1 ... minimal estimator
 // 2 ... minimum of jump and minimal estimator
 AposterioCemracs2007(
  Th,[u, dx(u), dy(u), rhs, u0, u0dx, u0dy, react],
  filename="filename",
  output=[etak[],etav[],etaDF[],etaR[],errk[],errv[],hd[],etaDF1[],etaDF2[]],
  medit=0,
  strategy=estimator);

  real exacterror = sqrt(errv[].sum); // exact error estimator
  real estDF = sqrt(etaDF[].sum);     // DF port of estimator
  real estR  = sqrt(etaR[].sum);      // R part of estimator
  real est   = sqrt(etav[].sum);      // estimated error

  cout << "Estimated error " << est << " Exact error " << exacterror << endl;

  // REGULAR GRID REFINEMENT
  Th=trunc(Th,1,split=2);
}
```

4.2 react.edp

Square domain problem, $\Delta u + ru = 0$.

```
// load cemracs module
load "cemracs2007";

// SQUARE PROBLEM
mesh Th=square(2,4,[x,y]);

// FE SPACES DEFINITION
fespace Vh(Th,P1); // the solution
fespace Pvh(Th,P1); // the indicator in each vertex
fespace Pth(Th,P0); // the indicator in each triangle

//  FE approximations
Vh u,v;
```

```
// EXACT SOLUTION DEFINITION
// reaction term
func react=10e6;
// exact solution function
func u0=exp(-sqrt(react)*x)+exp(-sqrt(react)*y);
// exact solution derivative du0/dx
func u0dx=-sqrt(react)*exp(-sqrt(react)*x);
// exact solution derivative du0/dy
func u0dy=-sqrt(react)*exp(-sqrt(react)*y);
// right hand side of the laplace equation (source term)
func rhs= 0;


// FE PROBLEM
problem Laplace(u,v)=
 int2d(Th)(u*v*react + dx(u)*dx(v) + dy(u)*dy(v))-int2d(Th)(rhs*v)
 +on(1,2,3,4,u=u0);


for (int i=0;i<10;i++) // refining the mesh
{
 u=u;
 Laplace;

 // ESTIMATOR VARIABLES
 Pvh etav;   // squares of final estimator (per vertex)
 Pvh etaDF;  // squares of DF part of the estimator
 Pvh etaR;   // squares of R part of the estimator
 Pvh errv;   // squares of computed error (per vertex)
 Pvh hd;     // computed mesh size
 Pvh etaDF1; // squares of DF(1) part of the est.
 Pvh etaDF2; // squares of DF(2) part of the est.
 Pth etak;   // squares of final estimator (per triangle)
 Pth errk;   // squares of computed error (per triangle)


 int estimator = 0;
 // 0 ... jump estimator
 // 1 ... minimal estimator
 // 2 ... minimum of jump and minimal estimator
 AposterioCemracs2007(
  Th,[u, dx(u), dy(u), rhs, u0, u0dx, u0dy, react],
  filename="filename",
  output=[etak[],etav[],etaDF[],etaR[],errk[],errv[],hd[],etaDF1[],etaDF2[]],
  medit=0,
  strategy=estimator);

  real exacterror = sqrt(errv[].sum); // exact error estimator
  real estDF = sqrt(etaDF[].sum);      // DF port of estimator
  real estR  = sqrt(etaR[].sum);       // R part of estimator
  real est   = sqrt(etav[].sum);       // estimated error

  cout << "Estimated error " << est << " Exact error " << exacterror << endl;

  // REGULAR GRID REFINEMENT
  Th=trunc(Th,1,split=2);
}
```

4.3 General usage

You need to have the cemracs2007.so in the same directory as the script files in order to execute diff.edp or react.edp scripts. Then, execute the scripts as follows

```
  FreeFEM++ diff.edp
  FreeFEM++ react.edp
```