CFD project
Mesh adaptivity for fluid flow over uncertain geometries

Vignesh Vittal-Srinivasaragavan

December 10, 2019

1 Introduction:

Adaptive meshes are very useful in the context of solving complex engineering problems as they offer the flexibility to use coarser mesh everywhere but at locations of interest and/or at locations of complex solution behavior. More accurate prediction of the solution can be computed with better efficiency using adaptive meshing than using finer mesh throughout the problem domain.

Mesh adaptation process typically involves computing the solution on a coarse mesh, and based on some error parameter, the locations of complex solution behavior is identified and the mesh is refined at these locations. When uncertainties are involved in a problem, it is not wise to demand the mesh to be refined adaptively every time the solution is computed (since the solution is computed numerous times for different realizations of the uncertain parameters). In this project, a method to identify an adapted mesh (which was adapted for some previous realization) and use it for a current realization (based on the compatibility of said adapted mesh for the current realization) is proposed. By eliminating the process of repeating the redundant adaptation steps, significant computational time is saved.

2 Problem statement:

A steady state incompressible flow with an uncertain advection at inlet in channel over a rectangular block of uncertain dimensions (aligned along the flow direction) is considered. Position of center of the rectangle is considered to be fixed

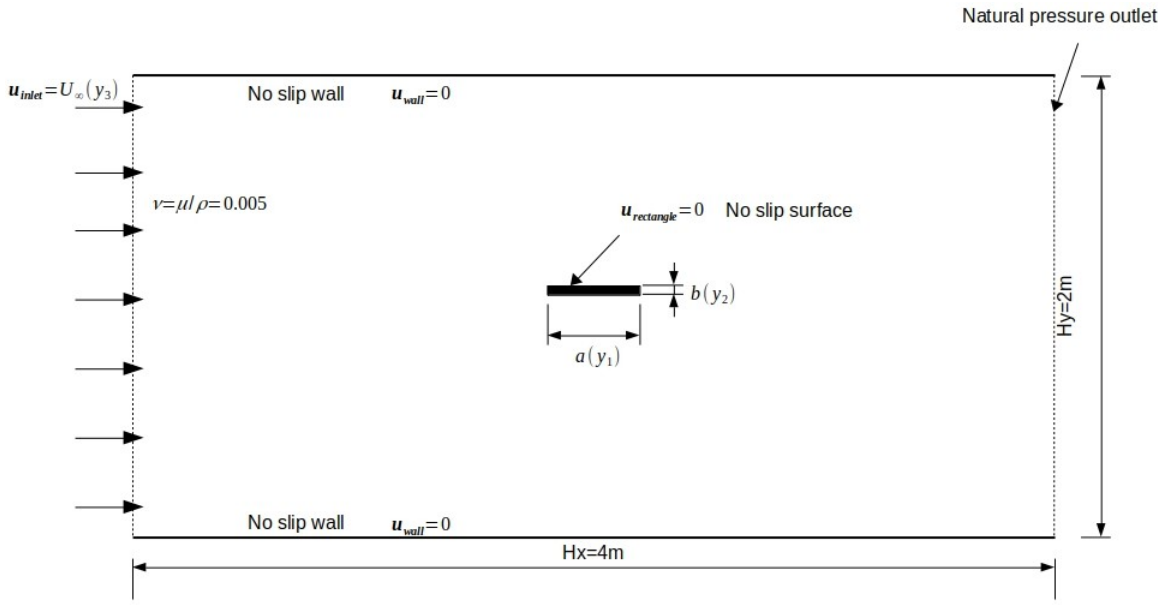


Figure 1: CFD problem setup with boundary conditions

The uncertain parameters are

- Length of the rectangular block

$$a = \mathcal{U}[350mm, 450mm] \implies a(y_1) = a_0 + 50y_1 \text{ with } a_0 = 400$$

- Width of the rectangular block

$$b = \mathcal{U}[20mm, 60mm] \implies b(y_2) = b_0 + 20y_2 \text{ with } b_0 = 40$$

- Fluid inlet velocity

$$\mathbf{u}_{inlet} = \mathcal{U}[5m/s, 10m/s]\hat{i} \implies U_\infty(y_3) = 7.5 + 2.5y_3$$

where y_is are the uniform random variable such that $y_i = \mathcal{U}[-1, 1]$. a_0, b_0 are the dimensions of mean geometry. The flow parameters are chosen keeping in mind that the flow will be laminar and steady state solution exist for the flow.

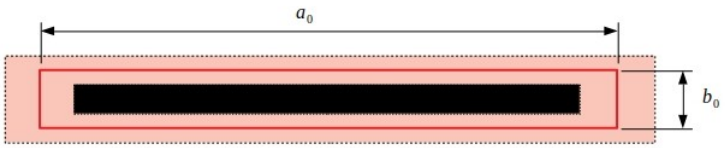


Figure 2: Dimensionally uncertain rectangular object. Shaded area around the solid red line (mean geometry) is the uncertainty in geometry

The quantity of interest are the expected drag force and its variance on the rectangular block in the channel.

3 Formulation:

3.1 Incompressible flow equations:

The fluid is modeled as an incompressible fluid which is governed by the following set of equations at steady state

$$\mathcal{R}^c = u_{k,k} = 0 \qquad \forall \mathbf{x} \in \Omega \tag{1}$$

$$\mathcal{R}^m_i = \rho u_j u_{i,j} + p_{,i} - \tau_{ij_{,j}} - f_i = 0 \quad \forall \mathbf{x} \in \Omega \tag{2}$$

where $\mathcal{R}^c$ and $\mathcal{R}^m_i$ are the strong form residual of the continuity (eq 1) and momentum equations (eq 2). $\tau = 2\mu\nabla^{\mathbf{S}}\mathbf{u} = 2\mu\left(\frac{1}{2}\left[\nabla\mathbf{u} + \nabla^{\mathbf{T}}\mathbf{u}\right]\right)$ is the viscous stress tensor. The strong form equations are supplemented with the boundary conditions to make the problem well posed. The boundary conditions are

$$\mathbf{u} = U_\infty \hat{i} \quad x \in \Gamma_{inlet} \quad \text{at inlet} \tag{3}$$

$$\mathbf{u} = \mathbf{0} \quad x \in \Gamma_{wall} \quad \text{no slip wall} \tag{4}$$

$$\mathbf{u} = \mathbf{0} \quad x \in \Gamma_{rect} \quad \text{no slip rectangular surface} \tag{5}$$

$$p = 0 \quad x \in \Gamma_{outlet} \quad \text{natural pressure outlet} \tag{6}$$

Finite element method is to be used to solve this CFD problem. Therefore, the Galerkin discrete weak form is given by

$$\begin{aligned} 0 = &\int_\Omega [-q_{,k}u_k + w_i\{\rho u_j u_{i,j} - f_i\} + w_{i,j}\{\tau_{ij} - p\delta_{ij}\}]d\Omega \\ &\int_{\Gamma\setminus\Gamma_g} q u_k n_k - w_i\{\tau_{ij} - p\delta_{ij}n_j\}d\Gamma \quad \forall w \in \mathcal{W}, q \in \mathcal{Q} \end{aligned} \tag{7}$$

$\mathcal{W}, \mathcal{Q}$ are the discrete test function spaces. The computed solution based on the Galerkin formulation is very unstable in the advection dominated flows, therefore, a stabilzed FE method is implemented whose discrete form is obtained by adding few stabiltity terms in the previous formulation as

$$\begin{aligned} 0 = &\int_\Omega [-q_{,k}u_k + w_i\{\rho u_j u_{i,j} - f_i\} + w_{i,j}\{\tau_{ij} - p\delta_{ij}\}]d\Omega \\ &+ \int_{\Gamma\setminus\Gamma_g} q u_k n_k - w_i\{\tau_{ij} - p\delta_{ij}n_j\}d\Gamma \\ &+ \sum_e \int_{\Omega_e} \left[\left(u_j w_{i,j} + \frac{q_{,i}}{\rho}\right)\tau_m \mathcal{R}^m_i + \tau_c w_{i,i}\mathcal{R}^c\right] d\Omega_e \\ &+ \sum_e \int_{\Omega_e} \left[w_i \rho \overset{\triangle}{u}_j u_{i,j} + \bar{\tau}\mathcal{R}^m_j w_{i,j}\mathcal{R}^m_k u_{i,k}\right] d\Omega_e \quad \forall w \in \mathcal{W}, q \in \mathcal{Q} \end{aligned} \tag{8}$$

with $\overset{\triangle}{u}_j = \frac{-\tau_m \mathcal{R}^m_j}{\rho}$. A set of non-linear equations are obtained when the trial and test functions are written in terms of linear shape functions, which upon solving the solution field is obtained and thereby the functional which in this case is the forces on the solid body.

3.2 VMS error estimator:

The spatial discretization is controlled with a posteriori error estimate. Hughes et al [1] introduced an analytical error estimator of 1D steady advection-diffusion problem which provides a pointwise exact error under certain conditions. Consider the advection-diffusion problem

$$\mathcal{R}(\phi) = a_i\phi_{,i} - \kappa\phi_{,ii} - f_i \tag{9}$$

In VMS method the solution is decomposed into an coarse scale solution $\bar{\phi}$, which is discretized in FE space and a fine scale solution ϕ', which can be used as a discretization error. Using the fine scale Green's function, the analytical form of the fine scale solution is evaluated as

$$\phi'(x_i, y_j) = \int_\Omega g'(x_i, y_j)\mathcal{R}(\bar{\phi})d\Omega \quad x_i, y_j \in \Omega \tag{10}$$

The VMS error computed at element level is described as the H1 semi-norm of the fine-scale solution

$$|\epsilon|_{H^1(\Omega_e)} = |\phi'_e|_{H^1(\Omega_e)} = |\mathcal{R}(\bar{\phi})| \left|\left| \int_{\Omega_e} g'_{,i} d\Omega_e \right|\right|_{L^2(\Omega_e)} \tag{11}$$

Using the analytical green's function it can be reduced to

$$|\epsilon|_{H^1(\Omega_e)} = ||\phi'_e||_{H^1(\Omega_e)} = \nu^{err}_{e,1D}||\mathcal{R}(\bar{\phi})||_{L^2(\Omega_e)} \tag{12}$$

where $\nu^{err}_{e,1D} = \frac{1}{|a_i|}\sqrt{Pe_e \coth(Pe_e) - 1}$ is the element error parameter and $Pe_e = \frac{h_e a_i}{2\kappa}$ is the cell Peclet number. The same can be extended to multi dimension advcetion diffusion problem using asymptotic approximations of the advective and diffusive limits as

$$\nu^{err}_e = \frac{1}{\sqrt{\kappa\sqrt{a_i g_{ij} a_j} + 3\kappa^2\sqrt{g_{ij}g_{ij}}}} \tag{13}$$

with $g_{ij} = \frac{d\xi_k}{dx_i}\frac{d\xi_k}{dx_j}$, while (eq 12) remains the same in multi-D problems. The incompressible Navier-Stokes equations possess similar advective and diffusive properties, which makes it easy to formulate its error estimator by extending the previously formulated VMS error estimator for advection diffusion problem

$$|\epsilon^{NS}|^2_{H^1(\Omega_e)} = \sum_{i=1}^{n_{sd}} |\epsilon^{NS}_i|^2_{H^1(\Omega_e)} \tag{14}$$

$$|\epsilon^{NS}_i|^2_{H^1(\Omega_e)} \approx \nu^{err,NS}_e||\mathcal{R}^m_i(\bar{u})||_{L^2(\Omega_e)} \tag{15}$$

$$\nu^{err,NS}_e = \frac{1}{\sqrt{\nu\sqrt{\bar{u}_i g_{ij} \bar{u}_j} + 3\nu^2\sqrt{g_{ij}g_{ij}}}} \tag{16}$$

Once the solution field is computed by a solver, the posterior element error is computed using (eq 14, 15, 16). The spatial discretization can then be controlled by specifying a target element size for elements with high estimated error. The element size with VMS error (when its defined as the H^1 norm of the fine scale solution) as

$$\left[\frac{h^{targ}_e}{h^{curr}_e}\right]^{1+\frac{n_{sd}}{2}} = \frac{\epsilon^{NS,targ}}{\epsilon^{NS,curr}} \tag{17}$$

4 Workflow for the CFD-UQ problem:

The entire workflow for the given uncertain flow problem has 5 major aspects –

- Initial meshing and mesh transformations
- Fluid Solve
- Error-estimation
- Mesh adaptation
- Quantify uncertainty

4.1 Initial Meshing and Mesh transformations:

Simmetrix software is to used for mesh related operations which in this case are primarily meshing and adapting the mesh. The initial mesh is created in simmterix, which can be done using simmetrix GUI or shell scripts.

Due to the uncertainties in the solid body's geometry, for every realization the mesh needs to be consistent with the problem domain for that realization. To circumvent the issue of creating a new geometry and meshing the same for every realization, a mesh-moving technique (developed in [2] and later tested on similar problem in [3]) is to implemented. In [2], the mesh movement is performed by treating the mesh as a linear elastic solid governed by constitutive relations. The mesh i.e. elastic body is then subjected to displacements at its boundaries (so that it transforms to the required geometry) and the displacement field for interior nodes are solved based on these BCs. However, for the sake of simplicity, the mesh movements in this project are prescribed using analytical transformations.

In this project, all the mesh related operations are only performed on the base domain which is considered as the domain that contains the base geometry (or mean geometry i.e the solid red line in figure 2). Mesh created on the base geometry (here on referred to as base mesh) will then be transformed to the actual domain of the problem using some transformation. The following figure explains this mesh transformation for a realization of solid geometry i.e $M_{(a,b)}$ from the base mesh $M_{(a_0,b_0)}$. Figure 3 shows the base and transformed coarse mesh which is to be used as the initial mesh that gets adapted for a given realization of the random variables. The base mesh is created in simmterix, however the transformed meshes are shown only for the purpose of visualization. The mesh transformations are not carried out simmterix.

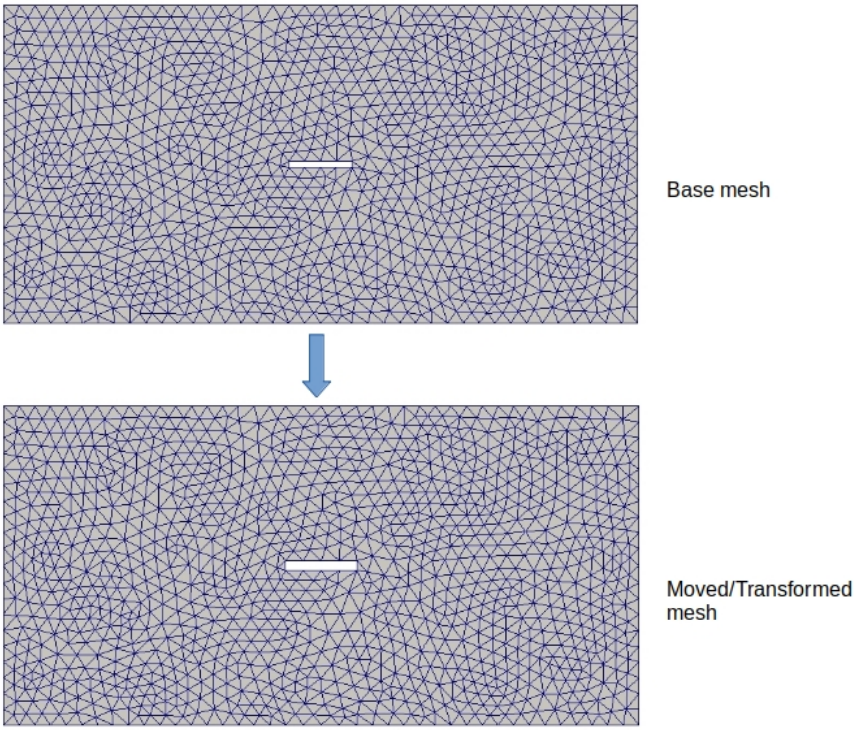


Figure 3: Mesh created on the base geometry (top–$M_{(a_0,b_0)}$) is transformed by specified displacements to the nodes consistent with the actual geometries to create the mesh at (bottom–$M_{(a,b)}$)

The reason for carrying out such transformation is that the mesh remains topologically same which allows one to make comparison between different meshes even though the domain geometry has changed.

Due to the linear nature of the problem geometry, the mesh transformations (also linear) from base to original geomtery can be analytically described by the following equations.

$$
\begin{aligned}
x^{trans} &= x^{base}\left(\frac{a}{a_0}\right) \qquad |x^{base}| \leq a_0/2 \\
x^{trans} &= x^{base} + \left(x^{base} - \frac{x^{base}}{|x^{base}|}\frac{H_x}{2}\right)\left(\frac{a-a_0}{a_0-H_x}\right) \quad |x^{base}| \geq a_0/2 \\
y^{trans} &= y^{base}\left(\frac{b}{b_0}\right) \qquad |y^{base}| \leq b_0/2 \\
y^{trans} &= y^{base} + \left(y^{base} - \frac{y^{base}}{|y^{base}|}\frac{H_y}{2}\right)\left(\frac{b-b_0}{b_0-H_y}\right) \quad |y^{base}| \geq b_0/2
\end{aligned} \tag{18}
$$

The displacements follow the constraints that the mesh nodes at the boundaries (wall and rectangular surface) can only slide along the boundary (it cannot go beyond or separate away from its respective boundaries). The transformation of mesh from base geometry to the actual problem geometry is carried out by changing coordinates of the nodes inside the CFD solver.

4.2 CFD Solver:

PHASTA CFD solver is to be used to compute the solution fields. Since the solver is written for transient flows, the problem will be solved until steady state is reached. In this case, the steady state is observed to have reached about after 2 seconds (for given BCs and static initial conditions). So for every solve, the solver computes the solution for 40 time steps which are 0.05s (Δt) apart from each other. The solution at $t = 2s$ is taken as the steady state solution.

The inputs to the solver are always the `geombc` and `restart` file corresponding to the base geometry. Mesh transformation is acheived through changing the nodal coordinates inside the `itrdrv()` in the `phSolver`. Once the mesh displacements are specified in `itrdrv()`, the solver will automatically compute the solution on the transformed mesh. However, it should be noted that the appropriate transformation needs to be performed before visualizing the solution on paraview, since the `geombc` file still corresponds to the base geometry.

The solution of a test case computed on the base mesh (for base geometry, therefore no transformations) for when $U_\infty = 7.5m/s, a = a_0, b = b_0$ using phasta's incompressible solver is shown below in figure 4.

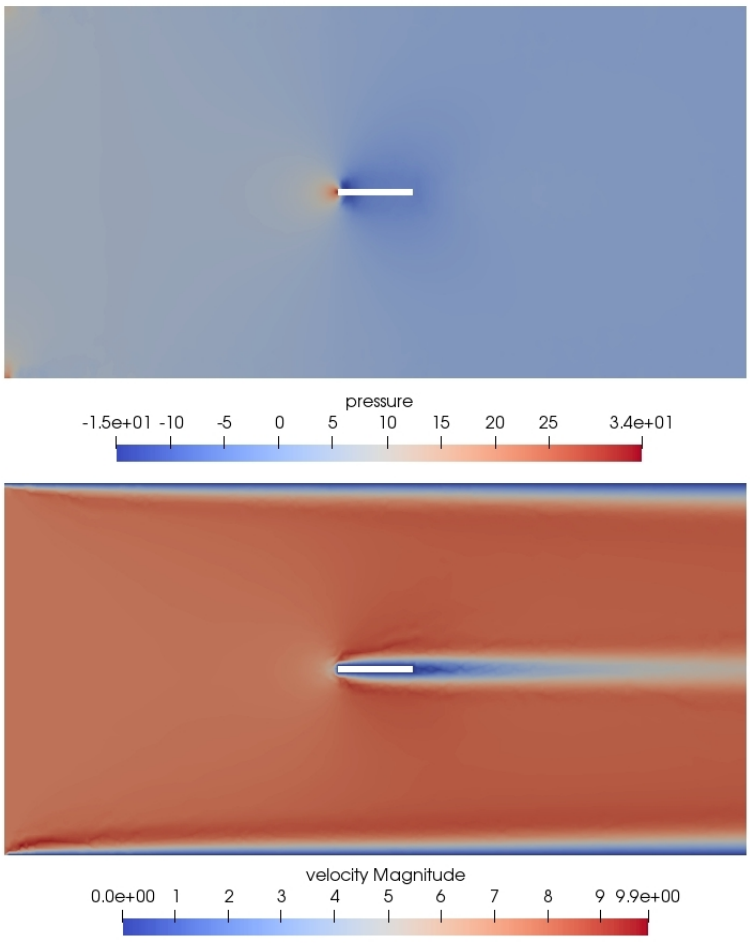


Figure 4: Pressure (top) and velocity magnitude (bottom) fields

NOTE:
Similarly, the values for uncertain inlet velocities are also realized by editing the velocity boundary conditions set in `phSolver/common/genBC1.f`. In the original model file, the velocity boumdary conditions are the inlet is specified a flag value – 2019 in this case. Then in the file `phSolver/common/genBC1.f`, an added script checks for all the nodes with velocity magnitude 2019, and then replaces it with the velocity of the realization.

4.3 VMS error-estimator:

Once the solution is computed on transformed coarse base mesh, the VMS error can be computed. However, for the purpose of comparing errors from different meshes, the solution field needs to be transformed back to base mesh and the error is estimated.

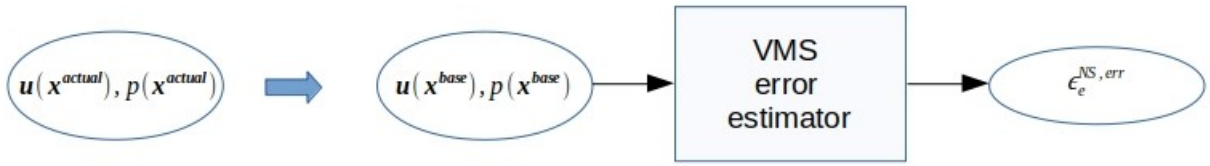


Figure 5: The solution is transformed to the base mesh and then the VMS error is estimated

A C++ code is to be built that reads the final solution field, perform the necessary transformation on to the base geometry, and then compute the element level VMS errors. Once the error field is computed, the simmetrix mesh adapt routine can be invoked if it is deemed necessary to adapt the mesh based on some error tolerance criteria.

A sample error field on a coarse mesh for the base geometry is shown in figure 6. For the purpose of presentation, MATLAB based script was used to compute the error field.

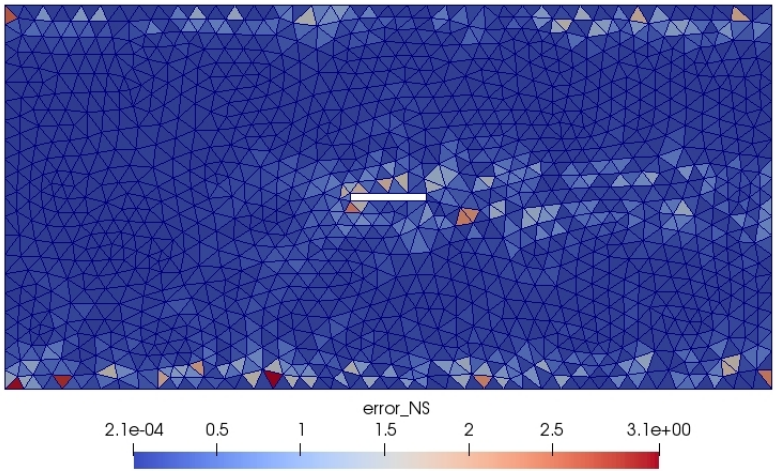
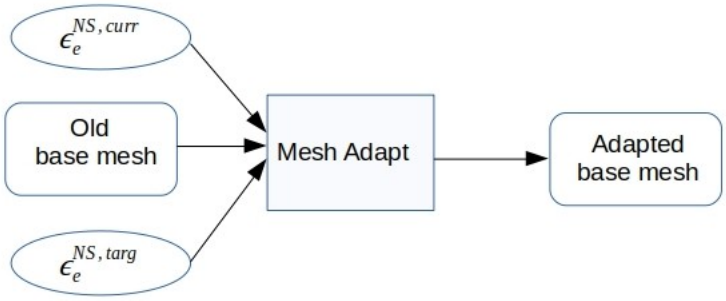


Figure 6: Element error field – $|\epsilon^{NS}|_{H^1_{\Omega_e}}$

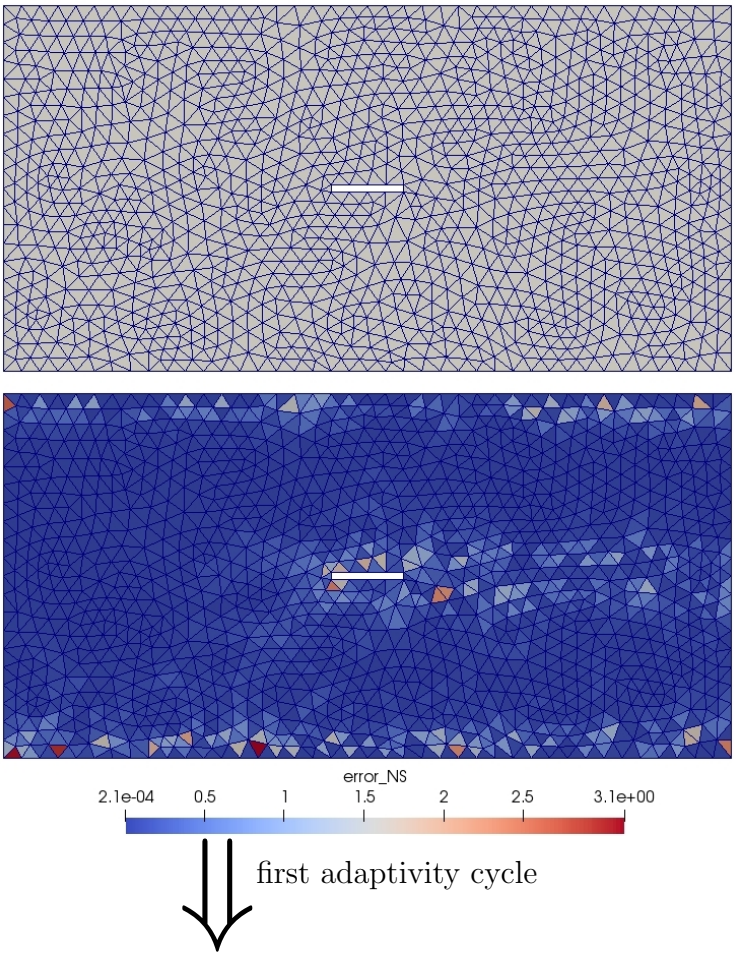
4.4 Mesh adaptation:

A code that uses the simmterix libraries and adapts a given mesh is ready and available. The code adapts the current mesh based on the error field calculated on that mesh.

When a mesh needs to be adapted for a given realization, the computed solution field is transformed on to the base mesh and then the error is estimated on this base mesh. The adaptation process is then carried out on the base mesh with the computed element error field. Once the base mesh is adapted, applying the same transformation, the adapted mesh for the actual geometry corresponding to the realization is obtained.



The following plots depict the adaptation process for two cycles of adaptivity starting from the base mesh used in the previous sections. The solution need not be transformed to compute the error estimator since the solution was computed on the base un-transformed mesh.



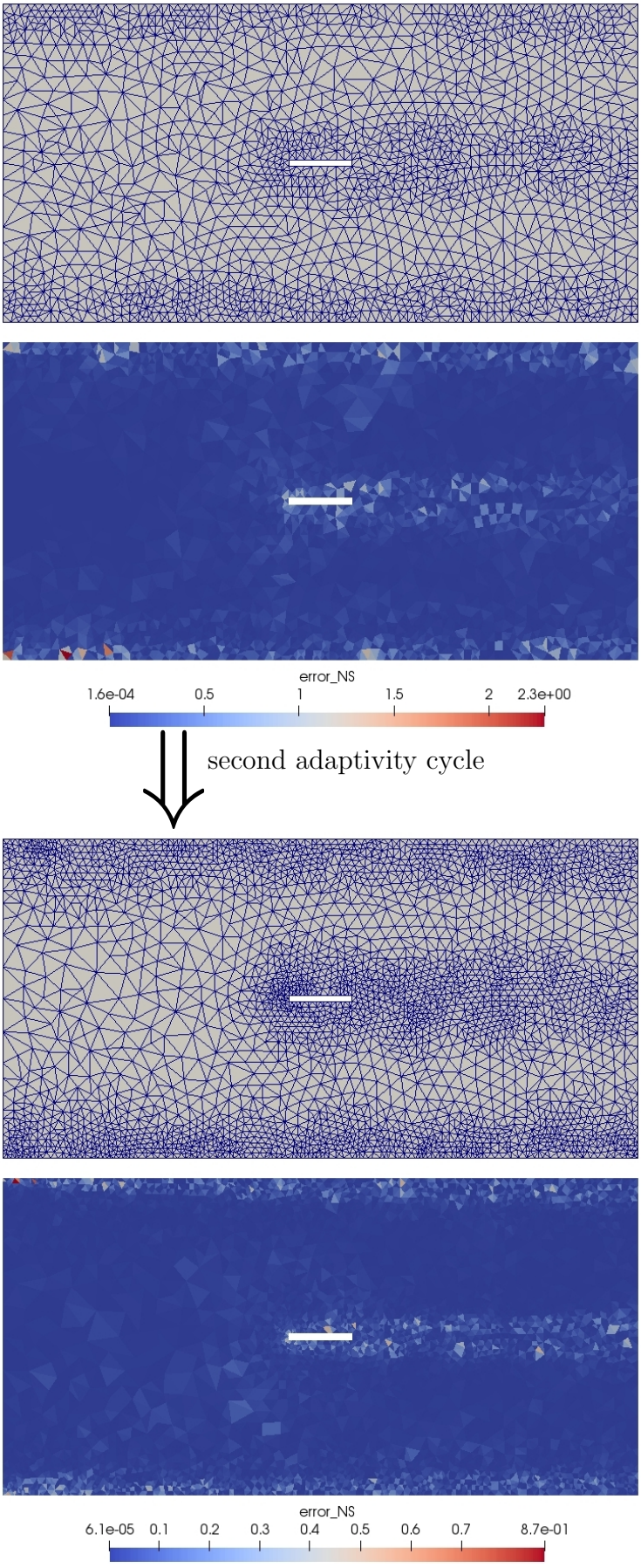


Figure 7: Mesh adaptation process based on computed error-field for two cycles of adaptivity

4.5 Quantifying Uncertainty:

Dakota UQ software will be used for the purpose of uncertainty quantification. It employs a non-intrusive generalized polynomial chaos based spectral methods to form a polynomial expression for the functional (drag/lift forces on the solid body in this case) based on computing the drag/lift forces for various instances of the uncertain parameters i.e realizations.

The higher the polynomial order for the functional expansion, the more accurate measure of the statistics obtained. As expected, depending on the polynomial order, the number of computations required to evaluate the polynomial also increases exponentially. Sparse grid based quadrature rules for integration of the polynomials and therefore computation of polynomial coefficients is to be employed. For 3 random variables, the following tables relates the sparse grid order, full polynomial order and the total number of integration points i.e total number of computations required to compute the coefficients of the polynomial.

n_{rv}	$\bar{l}$ (SG level)	$p^{\mathcal{S}}_{gpc}$ (total gpc order)	n_q^{tot} (integration points)
3	1	1	7
	2	2	31
	3	4	111
	4	6	351
	5	8	1023

It is clearly evident from the table that phasta solver needs to called numerous number of times in order to form a good polynomial approximation of the functionals. Therefore, adapting the mesh at each and every call will be unnecessary since there is a very high probability that the same adapted mesh will be obtained at multiple instances of solver runs. To avoid the redundant adaptations, a new method to reuse the adapted mesh based on comparing error field generated on the coarse mesh is introduced here.

4.5.1 Algorithm to re-use adapted mesh:

Consider a case where we have 2 levels of refinements (starting from a initial common coarse mesh i.e M0 mesh)

- For every realization, the error field on the M0 coarse mesh is computed. If the M0 error field does not match with a previously stored list of M0 error fields, then we add the computed error field to the list of error fields. Then, based on the computed (and stored) M0 error field, the elements are refined to obtain a new mesh (say M1 mesh). This process is repeated for one more cycle to obtain the final mesh M2. Therefore, every error-field on a M0 coarse mesh will correspond to final adapted mesh M2
- When the computed M0 error-field matches with a previously stored error-field (within a allowed tolerance), then the adapted mesh corresponding to that error-field will directly be used for the realization

The M0 error field for a realization i (say $\epsilon^{NS,err,M_0}_{e,(i)}$) is said to have matched the M0 error field of realization j (say $\epsilon^{NS,err,M_0}_{e,(j)}$), if for a specified tolerance value $\epsilon^{(tol)} << 1$ it satisfies

$$\frac{|\epsilon^{NS,err,M_0}_{e,(i)} - \epsilon^{NS,err,M_0}_{e,(j)}|_{L^2}}{\sqrt{N_{el,M_0}}} \leq \epsilon^{(tol)} \tag{19}$$

The following flowchart defines the overall workflow, for a given set of realization generated by the UQ driver (Dakota)

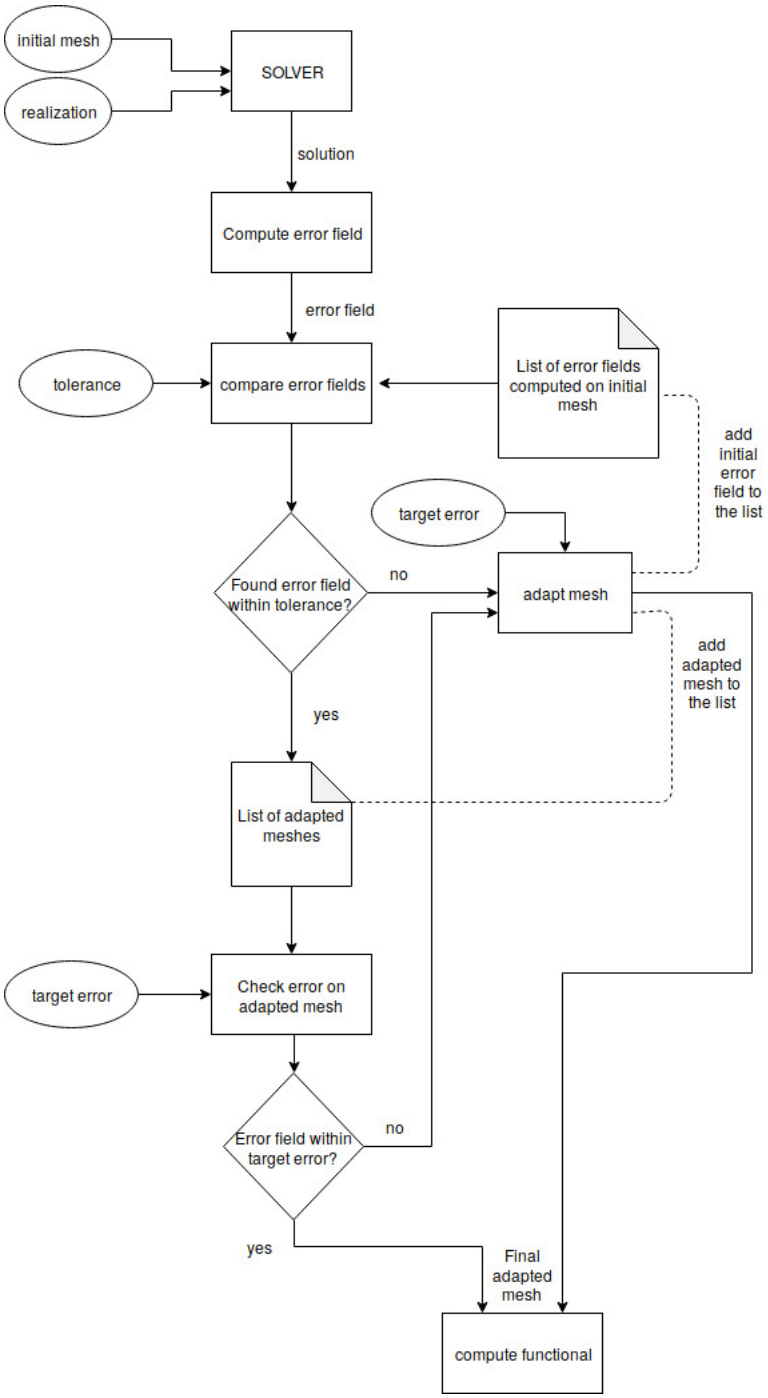


Figure 8: Overall workflow that incorporates the adaptive meshing algorithm

As mentioned before, for every realization the error computations are performed on

the solution field which is transformed to the base geometry. The mesh adaptation will then be carried out on the base geometry with the computed error field. The adapted final base mesh will then be transformed to the original geometry inside the phasta solver and used accordingly.

5 Code Development

Existing softwares:

- **Meshing**: Simmterix
- **CFD incompressible flow solver**: Phasta
- **UQ driver**: Dakota

Developed codes:

- **Mesh adaptivity code**: A code `adapt_mesh.cc` that uses simmterix libraries, reads a stored element error field and adapts a given mesh based on a specified element error tolerance (and other relevant mesh refine/coarsening parameters) is already developed and working.

  The code can be found here–

  ```
  /lore/vittav/PHASTA/CFD_UQ_project/rect_plate/mesh_scripts/
  adapt_mesh/adapt_mesh.cc
  ```

  NOTE: Mesh adaptation is performed in a 2D domain. This is possible because the mesh is created on a 2D geometry and extruded in normal direction to get the 3D mesh. In order to obtain the coordinate/connectivity details in 2D from a 3D initial mesh, a C++ code that uses simmetrix libraries is used.
  The code can be found here –

  ```
  /lore/vittav/PHASTA/CFD_UQ_project/rect_plate/mesh_scripts/
  initial_mesh/main.cc
  ```

  After the mesh is adapted, the new adapted mesh is obtained in 2D. Therefore, the 3D mesh (on which the PHASTA simulation is run) is obtained by once again extruding the 2D mesh in the normal direction. This is also achieved by a C++ code that uses simmetrix libraries.
  The code can be found here –

  ```
  /lore/vittav/PHASTA/CFD_UQ_project/rect_plate/mesh_scripts/
  extrude_mesh/main.cc
  ```

- **Problem variables in PHASTA**: The parameters that defines the geometry of domain (both deterministic and stochastic) and the one relating to uncertain velocity have to be declared and used inside PHASTA codes. 8 new variables

pertaining to the problem are added for this project. A new `struct` is declared in `/common/common_c.h` whose members are the 8 variables mentioned. The following snippets of the code are added to this file

```
extern struct {
int problem_flag;
double vinlet_curr;      // inlet velocity for current realization
double a0;                 // mean length of rect
double a_curr;             // length of rect for current realization
double b0;                 // mean width of rect
double b_curr;             // width of rect for current realization
double Hx;           // length of domain
double Hy;           // width of domain
} geomuq ;
```

Figure 9: `struct` where new variables are declared

```
#define sequence FortranCInterface_GLOBAL_(sequence, SEQUENCE)
#define amgvarr FortranCInterface_GLOBAL_(amgvarr, AMGVARR)
#define amgvari FortranCInterface_GLOBAL_(amgvari, AMGVARI)

#define geomuq FortranCInterface_GLOBAL_(geomuq, GEOMUQ)

#define MAXBLK   50000
#define MAXSURF  20
#define MAXTS    100
#define MAXTOP   5
#define MAXQPT   125
#define MAXSH    125
```

Figure 10: Creates a copy `struct` and its variables to be used in Fortran

The file `common.h` contains the common blocks and the data declaration needed for the routines. The highlighted line is added in `common_c.h`

```
      common /timdat/ time,   CFLfld, CFLsld, Dtgl,   Dtmax,  alpha,
     &                etol,   lstep,  ifunc,  itseq,  istep,  iter,
     &                nitr,   almi,   alfi,   gami,   almiS,   alfiS,
     &                gamiS, flmpl,  flmpr, dtol(2),  rtimevalue,
     &                itimevalue,    iCFLworst
c
    common /geomuq/ problem_flag, vinlet_curr, a0, a_curr, b0, b_curr, Hx, Hy
c
      common /timpar/ LCtime, ntseq
c
      common /incomp/ numeqns(100), minIters, maxIters,
     &                iprjFlag,     nPrjs,    ipresPrjFlag, nPresPrjs,
     &                prestol,      statsflow(6), statssclr(6),
     &                iverbose
c
      character*8     ccode
      common /mtimer1/ ccode(13)
```

Figure 11: Input variables that have been previously declared in `common_c.h` have to be re-declared here, in a consistent block fashion

For the current problem, the problem flag is declared as 2019 in `solver.inp` file. In the `/common/input_fform.cc` the values for other 7 variables is read from the file `rv_input.txt` (which is generated by DAKOTA for each realization) as –

```
if ( (int)inp.GetValue("Problem Flag") == 2019 ) {
  std::ifstream rv_inp_file("rv_input.txt");
  if (rv_inp_file.is_open()){
    rv_inp_file >> geomuq.vinlet_curr;
    rv_inp_file >> geomuq.a0;
    rv_inp_file >> geomuq.a_curr;
    rv_inp_file >> geomuq.b0;
    rv_inp_file >> geomuq.b_curr;
    rv_inp_file >> geomuq.Hx;
    rv_inp_file >> geomuq.Hy;
    rv_inp_file.close();
  }
}
```

Figure 12: Reading the inputs from `rv_inputs.txt`

For reference, the file `rv_inputs.txt` is created from `rv_inputs.template` which is in the following format (*comments in red are added for clarity, the actual file do not contain any comments*) –

```
{x1}      %inlet velocity for current realization--vinlet_curr
0.4                              %mean length of rectangle--a0
{x2}      %length of rectangle for current realization--a_curr
0.04                              %mean width of rectangle--b0
{x3}       %width of rectangle for current realization--b_curr
4.0                                        %length of domain--Hx
2.0                                         %width of domain--Hy
```

The values for `x1,x2,x3` are replaced by Dakota (in `rv_input.txt`) for each realization with appropriate values.

The modified codes (to read the inputs) can be found here –

```
/lore/vittav/PHASTA/phasta_SCOREC_CFDUQ/phSolver/common/common_c.h
/lore/vittav/PHASTA/phasta_SCOREC_CFDUQ/phSolver/common/common.h
/lore/vittav/PHASTA/phasta_SCOREC_CFDUQ/phSolver/common/input_fform.cc
```

- **Mesh transformation**: The phasta solver is modified to transform the base mesh coordinates (read from `geombc` file) to actual geometry based on the values of a and b for the given realization. The mesh transformation is acheived by prescribing displacements to the base mesh nodes accordingly as mentioned in (eq 18). In phasta, the transformation is done inside `itrdrv()` as

```
c...    mesh movement parameters
    my_above = (b_curr-b0)/(b0-Hy);
    my_below = (b_curr-b0)/b0;
    mx_above = (a_curr-a0)/(a0-Hx);
    mx_below = (a_curr-a0)/a0;
c...    prescribe new nodal coords
    where (abs(xo(:,1)).lt.(a0/2))
      xo(:,1) = xo(:,1) + xo(:,1)*mx_below
    endwhere
    where (xo(:,1).ge.(a0/2))
      xo(:,1) = xo(:,1) + (xo(:,1)-(Hx/2))*mx_above
    endwhere
    where (xo(:,1).le.-(a0/2))
      xo(:,1) = xo(:,1) + (xo(:,1)+(Hx/2))*mx_above
    endwhere

    where (abs(xo(:,2)).lt.(b0/2))
      xo(:,2) = xo(:,2) + xo(:,2)*my_below
    endwhere
    where (xo(:,2).ge.(b0/2))
      xo(:,2) = xo(:,2) + (xo(:,2)-(Hy/2))*my_above
    endwhere
    where (xo(:,2).le.-(b0/2))
      xo(:,2) = xo(:,2) + (xo(:,2)+(Hy/2))*my_above
    endwhere
```

The modified `itrdrv()` codes can be found here –
`/lore/vittav/PHASTA/phasta_SCOREC_CFDUQ/phSolver/incompressible/itrdrv.f`

- **Uncertain velocity**: The file `phSolver/common/genBC1.f` is edited to set the value of U_∞ for a given realization. The field values at boundary conditions comes from `geombc` file which is generated from the simmterix's `.smb` (model file) and `.sms` (mesh file) files. Since the model file is created on the base mesh, the inlet velocity will be specified in the model file as the expected inlet velocity i.e $U_\infty = 7.5m/s$. This value will be rightly perturbed (based on the value of y_3) as $U_\infty = 7.5 + 2.5y_3$ for a given realization inside `genBC1()`

```
c.......If magnitude of velocity is equal to inlet bc flag value
c   edit velocity to velocity of current realization
c...    if (problem_flag.eq.2019) then
      where (abs(BCtmp(:,7)-2019.0).lt.0.00001)
        BCtmp(:,7) = vinlet_curr
      endwhere
c...    endif

        if (nsd .eq. 3) then
        where (ibits(iBC,3,3) .eq. 7)
          BC(:,3) = BCtmp(:,7) * BCtmp(:,4)
          BC(:,4) = BCtmp(:,7) * BCtmp(:,5)
          BC(:,5) = BCtmp(:,7) * BCtmp(:,6)
        endwhere
        endif
```

The modified `genBC1.f` file can be found here –
`/lore/vittav/PHASTA/phasta_SCOREC_CFDUQ/phSolver/common/genBC1.f`

- **VMS error estimation**: A C++ code (`estimate_error.cpp`) that reads the solution field, transforms it to base mesh and then computes the element error field

is developed. Another c++ code was also developed that goes through the list of M0 error fields and return the number of matched error field. If no error field is matched, then the computed M0 error field will be added to the list and adapted after.

`/lore/vittav/PHASTA/CFD_UQ_project/rect_plate/gen_scripts/VMS_error.cpp`

- **Dakota driver script**: A shell script (`dakota_driver_reuse.sh`) that is executed by dakota to run a particular realization is developed. This is the script that couples the UQ driver (Dakota) to the solver (Phasta). The entire workflow depicted by the flowchart in previous section is realized through this script.

  The script can be found here –
  `/lore/vittav/PHASTA/CFD_UQ_project/rect_plate/gen_scripts/dakota_driver_reuse.sh`

Other shell scripts that were used in the process of mesh adaptation, error comparison to find matched mesh and such can be found here –

`/lore/vittav/PHASTA/CFD_UQ_project/rect_plate/gen_scripts`

The `readme.txt` file in the same folder contains the details of each script or code file present

6 Results and Conclusions

In the tested cases, two cycles of adaptivity are performed from the coarse mesh to obatain the final mesh. However, for a given realization, if the error on the C_0 mesh matches with previously adpated realization, that adapted mesh is direclty considered. This basically eliminates computing the solution on C_1 level meshes. For reference, C_0 mesh has 2252 elements (same for all realizations); C_1 mesh has around 3500 elements and C_2 mesh has about 6300 elements.

The expectation and variance of the drag force is computed using the polynomial chaos model. Sparse grid level 3 and 4 quadrature rules were implemented and results are tabulated in the following tables –

l_{SG}	match tol	reused mesh		Drag Force [N]		% Error	
		count	%	$\mathbb{E}$	$\mathbb{V}$	$\mathbb{E}$	$\mathbb{V}$
3	**0.0**	**0**	**0%**	**3.9813097239e − 02**	**3.0793907455e − 02**	–	–
	0.006	51	46%	4.1183857818e-02	3.1201013825e-02	3.44%	1.32%
	0.005	37	33%	3.9842328124e-02	3.2039922018e-02	0.07%	4.04%

l_{SG}	match tol	reused mesh		Drag Force [N]		% Error	
		count	%	$\mathbb{E}$	$\mathbb{V}$	$\mathbb{E}$	$\mathbb{V}$
4	**0.0**	**0**	**0%**	**4.4621994998e − 02**	**9.1298751794e − 04**	–	–
	0.005	193	55%	4.5927465735e-02	9.8783490913e-04	2.92%	8.1%
	0.004	160	45%	4.5450880554e-02	1.0149140082e-03	1.85%	11.16%
	0.003	114	32%	4.4927919289e-02	9.3686751794e-04	0.68%	2.61%

The following observations can be made from the tabulated results –

- In both table the each row is compared against the ones in bold (which is the case where for all realization the mesh was adapted). This is the best estimation of the QoI for a given level of sparse grid
- Reducing the matching tolerance reduces the number of realizations which reuses a mesh. However, the estimated expectation value is more accurate
- The estimated variance do not show any pattern, primarily because accurate predictions of variance comes from higher order approximation of the QoI

The results suggest that the algorithm performs really well while estimating the expected drag force. However, the method is not entirely robust. There needs to be a way to find the optimum match tolerance that saves significant computations while not compromising the accuracy of estimated QoIs. This needs to be further investigated on. The mesh transformation step needs to be more robust too. Cases with large range in geomteric uncertainty can often result in poorly deformed elements. In [2] (where the displacements are obtained by solving the linear elastic model of the mesh) this is adressed by altering the Jacobian (in the weak-form integral) using a non-negative stiffening power χ. Each element is stiffened by a factor $(J^e)^{-\chi}$ and χ determines the degree by which the smaller elements are rendered stiffer than the larger ones. Another thing to study further is check how often a matched mesh in C_0 level end up being a poor fit for a realization. In that case, it is possible to have a tree like structure where subsequent level meshes can be compared against if multiple meshes matched with a single mesh but only one ends up being a good fit.

The results are stored in dakota output files which can be found here –
`/lore/vittav/PHASTA/CFD_UQ_project/rect_plate/geomUQ_v2/results`

The `readme.txt` file in `/geomUQ_v2` folder contains the details on how to run a dakota simulation. Dakota has to installed in order to run the simulation.

References:

[1] Hughes, T.J. & Feijo, G.R. & Mazzei, L. & Quincy, J.B. (1998). The variational multiscale method – a paradigm for computational mechanics. Computer Methods in Applied Mechanics and Engineering. 166. 3-24.

[2] Stein, K. & Tezduyar, Tayfun & Benney, R. (2003). Mesh Moving Techniques for Fluid-Structure Interactions with Large Displacements. Journal of Applied Mechanics. 70. 58-63. 10.1115/1.1530635.

[3] Skinner, Ryan & Doostan, Alireza & Peters, Eric & Evans, John & Jansen, Kenneth. (2019). Reduced-Basis Multifidelity Approach for Efficient Parametric Study of NACA Airfoils. AIAA Journal. 57. 1-11. 10.2514/1.J057452.