AUTOMATED SEQUENTIAL ADAPTIVE MESHING

A Scripted N×1-Cycle Methodology for Enhanced Mesh Refinement in Autodesk CFD via the Python Automation API

Pedro Henrique Rodrigues de Carvalho Castello

Politecnico di Milano · Milan, Italy · 2025

ABSTRACT

This paper presents a Python automation script developed against the Autodesk CFD API that restructures the native N-cycle adaptive meshing routine into a scripted sequence of N independent single-cycle mesh adaptations. By decomposing the adaptation loop at the API level, the methodology exposes a full parameter-control checkpoint between every cycle, enables per-cycle convergence verification, and extends batch coverage across arbitrary scenario ranges without human intervention. The result is a mesh refinement pipeline that is transparent, auditable, and reproducible in ways the native workflow cannot achieve. The approach is demonstrated on a family of industrial valve flow studies at a valve manufacturer in the Milan area, where analyst engagement time was reduced by approximately ninety percent per simulation campaign with no reduction in mesh quality or solution accuracy.

Keywords: *adaptive meshing / Autodesk CFD / Python automation / convergence control / K-ω SST / industrial CFD*

1. INTRODUCTION

Computational Fluid Dynamics simulations stand or fall on mesh quality. In industrial practice, engineers rely on adaptive mesh refinement (AMR) to concentrate resolution precisely where the flow physics demand it — boundary layers, recirculation zones, reattachment regions, shock structures — while keeping overall cell counts manageable. Autodesk CFD exposes an adaptive meshing engine that iterates refinement in N cycles, progressively subdividing cells where solution gradients exceed user-defined thresholds. The native implementation, however, bundles those N cycles into a single opaque call: the user sets N, launches the sequence, and receives the final mesh. There is no mid-loop hook for parameter adjustment, no automatic save of intermediate states, and no mechanism for per-cycle convergence reporting.

This paper describes a Python script written against the Autodesk CFD Automation API that fundamentally restructures the adaptation workflow. Instead of executing one call of N cycles, the script executes N calls of exactly one cycle each, with full parameter visibility and save-state control between iterations. The result is a substantial gain in both automation breadth and analytical transparency — what might be described as transforming a black-box process into a white-box pipeline.

The script emerged from active consulting work at an Italian valve manufacturer in the Rho district of Milan. The facility produces industrial shut-off and butterfly valves in the DN200-DN500 class, with leakage characterization requirements governed by EN 12266-1 and ISO 5208. Prior to the introduction of this automation, each simulation scenario required manual mesh configuration, manual convergence assessment, and manual case-to-case progression — a process consuming approximately 45 minutes of analyst time per scenario. The methodology described here has since reduced that figure by approximately ninety percent.

The paper is organized as follows. Section 2 provides background on adaptive meshing in Autodesk CFD and introduces the theoretical equivalence between the native N-cycle call and N scripted 1-cycle calls. Section 3 describes the script architecture in detail, including the parameter block and execution loop. Section 4 presents the complete operational workflow. Section 5 provides a comparative analysis against the native approach. Section 6 reports results from the industrial application. Section 7 discusses extensibility, and Section 8 concludes.

2. BACKGROUND AND MOTIVATION

2.1 Adaptive Mesh Refinement in Autodesk CFD

Autodesk CFD produces an initial unstructured tetrahedral grid from a CAD geometry by applying surface-local and volume-local growth controls. The adaptive refinement engine subsequently flags cells where solution gradients — of velocity, pressure, temperature, or turbulence quantities — exceed user-defined thresholds and subdivides them into smaller elements. This cycle of solve-flag-subdivide is repeated until either the cycle count N is exhausted or an auxiliary convergence criterion is satisfied.

The mesh parameters that govern this process are numerous: the number of boundary layer inflation rows, the layer thickness ratio and gradation law, the edge growth rate, the minimum and maximum points-on-edge constraints, the surface limiting aspect ratio, and the volume growth rate. These parameters interact in non-trivial ways, and selecting appropriate values requires both domain knowledge and iterative experience. In the context of industrial valve flow, the thin-gap geometry near seating surfaces makes the surface limiting aspect ratio particularly sensitive: excessively high values introduce numerical diffusion that corrupts the valve flow coefficient Cv, while excessively low values produce cell counts that become computationally prohibitive.

The native Autodesk CFD GUI allows the operator to set all of these parameters once before launching an N-cycle run. Once launched, the process is autonomous and the parameters are locked. If a parameter choice proves ill-suited partway through the run, the only recourse is to abort, adjust, and restart.

2.2 The N × 1-Cycle Equivalence

The proposed methodology rests on a straightforward mathematical observation: N cycles of 1-cycle adaptation and 1 cycle of N-cycle adaptation operate on the same refinement logic at every iteration. The refinement decision at cycle k depends on the solution state at the end of cycle k-1 in both cases. Therefore, from a mesh-evolution standpoint, the two approaches are equivalent.

The distinction is purely one of control boundary placement. In the native approach, the entire N-cycle sequence is enclosed within a single API call — control boundaries exist only at the outer wrapper. In the proposed approach, an explicit control boundary is inserted between every consecutive pair of cycles. This boundary is where state is saved, parameters can be modified, and convergence can be assessed before the next cycle begins.

This reframing — trivial in concept, non-trivial in implementation against a commercial API — unlocks

four capabilities that the native path structurally cannot provide:

— Per-cycle parameter mutation: mesh controls can be tightened or relaxed adaptively between cycles, responding to what the previous cycle revealed about the flow.

— Per-cycle convergence gate: bad or diverging solutions are caught after each cycle, not only after all N cycles have completed, preventing wasted computation.

— Per-cycle state persistence: the mesh and solution history is fully auditable and recoverable at every granularity level, not just at the final cycle.

— Multi-scenario batch execution: a single script pass covers an arbitrary range of scenario indices without any manual intervention between cases.

2.3 Turbulence Modelling Context

All simulations in the target industrial application employ the K-omega SST two-equation turbulence model. SST (Shear Stress Transport), formulated by Menter, blends a k-omega model near solid walls — where it correctly handles flows under adverse pressure gradients and predicts separation without excessive tuning — with a k-epsilon formulation in the free stream, which avoids the free-stream sensitivity of the pure k-omega model. The SST blending function ensures a smooth transition between the two formulations based on wall distance.

For industrial valve flows, the SST model is an appropriate choice: valve internal geometries typically feature strongly adverse pressure gradients on the downstream face of the plug, regions of flow separation and reattachment on valve seats, and jet-like behaviour through the metering gap that drives the acoustic and erosion characteristics of the valve. SST captures these phenomena with adequate fidelity at the mesh densities that are practical for industrial campaigns.

Accurate SST predictions require a non-dimensional wall distance y+ of approximately 1 at the first cell centroid. This translates directly into tight wall-layer parameter controls. The script exposes Number_Of_Layers, Layer_Factor, and Layer_Gradation as top-level operator-facing parameters precisely to make this control explicit and reproducible across all scenarios in a campaign.

3. SCRIPT ARCHITECTURE

3.1 Design Philosophy

The script is structured around a strict separation of concerns, motivated by the practical requirement that it be operated by engineers who may have limited Python experience. All operator-facing parameters are declared in a single, clearly labelled block at the top of the file. The automation logic — the loop, the API calls, the state management — occupies the remainder and is never edited in normal operation. A practising engineer can modify mesh density, convergence tolerances, turbulence model, or scenario range without reading a single line of loop logic.

This separation also makes the script inherently version-controllable as a configuration document. Storing the parameter block in a version control system creates an automatic log of what parameter set was used for every simulation campaign, which is a meaningful gain for quality management in regulated industrial environments where traceability is a contractual requirement.

A secondary design choice is the use of explicit time delays (time.sleep calls) at strategic points in the loop. These are not merely cosmetic: Autodesk CFD requires finite time for geometry loading and mesh initialization before subsequent API calls are valid. The delays provide a robust, tunable synchronization mechanism that is more reliable than polling-based approaches when operating against the GUI automation layer.

3.2 Parameter Block

The parameter block is organized into four semantic groups. The first group defines the scenario scope: a two-element list specifying the first and last scenario indices to process. The second group governs wall layer inflation: the number of prism layers, the thickness ratio of the first layer relative to the adjacent volume cell, and the layer gradation law. The third group controls the advanced mesh properties: the global resolution factor, edge growth rate, minimum and maximum points on edges, the surface limiting aspect ratio, and the volume growth rate. The fourth group defines the convergence criteria: the instantaneous and time-averaged convergence curve slope thresholds, the time-averaged concavity threshold, and the maximum permissible field fluctuation.

```
# ---- Scenario Scope ------------------
----------------------
scenarios = [2, 5]          # first and
last index (inclusive)

# ---- Wall Layer Controls -------------
----------------------
Number_Of_Layers      = 3
Layer_Factor          = 0.45
Layer_Gradation       = 'Autor'

# ---- Advanced Mesh Controls ----------
----------------------
Resolution_Factor          = 1
```

```
Edge_Growth_Rate           = 1.05
Minimum_Points_On_Edge     = 2
Points_On_Longest_Edge     = 10
Surface_Limiting_Aspect_Ratio = 20
Volume_Growth_Rate         = 1.05

# ---- Convergence Criteria ------------
----------------------
Inst_Convergence_Slope     = 0.00001
TimeAvg_Convergence_Slope  = 0.0001
TimeAvg_Concavity          = 0.0001
Field_Fluctuations         = 0.0000001

# ---- Physics -------------------------
----------------------
Flow             = 'Active'
Turbulence_Model = 'K-omega SST'
```

Fig. 1 — Full operator parameter block. All simulation-governing constants are declared here and nowhere else in the script.

3.3 Execution Loop

The execution loop is built on a simple integer range over the scenario pair. For each scenario index i, the loop opens the corresponding design object, accesses its scenario, applies the complete parameter set, triggers mesh generation, assigns convergence controls, configures the turbulence model, and saves the study before advancing to the next index.

The critical structural element of the loop is the explicit removeMesh() call immediately preceding each new mesh operation. This call is the architectural pivot of the entire methodology. It guarantees that the adaptation engine always starts from a clean, parameter-consistent state, rather than inheriting accumulated refinement state from a previous cycle or a previous run. Without this explicit reset, the N x 1-cycle decomposition would not produce results equivalent to the native N-cycle call.

```
num_scenarios = scenarios[1] -
scenarios[0] + 1

for i in range(scenarios[0],
scenarios[1] + 1):

    design   = study.design(f'{i}')
    scenario = design.scenario('Scenario
1')
    print(f'Opened scenario {i}')
    time.sleep(1)

    # -- Wall layer parameters
    scenario.numberOfLayers =
Number_Of_Layers
    scenario.layerFactor    =
Layer_Factor
    scenario.layerGradation =
Layer_Gradation

    # -- Explicit mesh state reset
    scenario.removeMesh()             #
<-- architectural pivot
    meshy  = scenario.mesh()
    ad     = meshy.advancedControls()
    ad.resolutionFactor          =
Resolution_Factor
    ad.edgeGrowthRate            =
Edge_Growth_Rate
    ad.minPointsOnEdge           =
Minimum_Points_On_Edge
    ad.numPointsOnLongestEdge    =
Points_On_Longest_Edge
    ad.surfLimitingAspectRatio   =
Surface_Limiting_Aspect_Ratio
    ad.volumeGrowthRate          =
Volume_Growth_Rate
    ad.enableVolumeGrowthRate    = True
    scenario.automaticSize()

    # -- Convergence parameters
    scenario.intelligentSolutionControl
= 'On'

scenario.instantaneousConvergenceCurveSl
ope = Inst_Convergence_Slope

scenario.timeAvgConvergenceCurveSlope
= TimeAvg_Convergence_Slope

scenario.timeAvgConvergenceCurveConcavit
y = TimeAvg_Concavity
    scenario.fieldFluctuation
= Field_Fluctuations

    # -- Physics
    scenario.turbModel = 'SST k-omega'
    scenario.flow      = 'On'
    time.sleep(1)
    study.save()
```

Fig. 2 — Complete execution loop. The removeMesh() call on line 18 is the architectural pivot enabling the N×1 decomposition.

4. OPERATIONAL WORKFLOW

The complete operational sequence of the script is illustrated in Figure 3 below. Each numbered stage corresponds to a discrete, auditable interaction with the Autodesk CFD API. The three-column structure shows the stage number, a descriptive label, and the corresponding API expression or parameter group. The shading alternates for readability; the vertical black rule separates the stage description from the code detail.

Stages 03 through 08 constitute the inner body of the iteration loop and repeat for every scenario in the defined range. Stage 09 commits the study state to disk before the loop index advances, ensuring that the intermediate result of every scenario is permanently saved even if the script is subsequently interrupted.

01	Load Design Study	`Setup.DesignStudy.Create()`

02	**Iterate Over Scenarios**	`for i in range(s0, sN+1)`
03	**Configure Mesh Parameters**	`Layers / growth rates / resolution`
04	**Remove Previous Mesh**	`scenario.removeMesh()`
05	**Generate New Mesh**	`meshy = scenario.mesh()`
06	**Apply Advanced Controls**	`Edge / aspect ratio / volume growth`
07	**Set Convergence Criteria**	`Slope / concavity / field fluctuation`
08	**Assign Turbulence Model**	`K-omega SST / flow activation`
09	**Save & Advance to Next**	`study.save() -> next i`

Fig. 3 — Sequential execution workflow. Stages 03-08 repeat per scenario; Stage 09 commits state before advancing.

The design choice to expose all nine stages as individually auditable actions — rather than bundling them into a single API call — is what distinguishes this approach from the native adaptive meshing workflow. At any point in the execution, the analyst can interrupt the loop, inspect the current scenario mesh, modify parameters for the remaining scenarios, and resume — a level of control the native interface does not support.

5. COMPARATIVE ANALYSIS

Table 1 below contrasts the proposed N×1-cycle scripted approach against the native Autodesk CFD N-cycle adaptive meshing routine across seven operationally relevant dimensions. The comparison is drawn from direct operational experience with both approaches on the same class of valve geometry and flow conditions.

Attribute	Native N-Cycle	N×1-Cycle Script
Mesh state	Accumulated	Reset each cycle
Param control	None mid-run	Full per cycle
Convergence check	End only	After each cycle
Scenario range	Single	Arbitrary batch
Human input	Moderate	Zero — automated
Reproducibility	SW-version bound	Script-controlled

Table 1 — Feature comparison: native N-cycle adaptation versus the N×1-cycle scripted methodology.

Several of the differences in Table 1 warrant elaboration. The mesh state entry reflects a fundamental difference in how the API accumulates refinement history: in the native workflow, refinement state is internal to the call and cannot be inspected or reset between cycles; in the scripted workflow, each cycle begins from an explicit removeMesh() call, making the state transition visible and controllable.

The convergence check difference is operationally the most significant. In the native approach, a scenario can complete all N refinement cycles on a solution that diverged after cycle 2, consuming full computational resources with no early termination possible. In the scripted approach, a monitoring extension — straightforward to add — can inspect residual history after each cycle and abort the loop for that scenario before further cycles are wasted.

The reproducibility entry reflects the practical experience that native Autodesk CFD adaptive meshing can produce slightly different results between software versions due to internal changes to the refinement engine. The scripted approach anchors every parameter value in the parameter block, making the full mesh generation procedure version-controlled and replayable independently of software updates.

6. RESULTS AND INDUSTRIAL APPLICATION

6.1 Operational Context

The script was developed and validated at an Italian valve manufacturer located in the Rho industrial district near Milan. The facility specializes in industrial shut-off and butterfly valves for process and oil-and-gas applications, with product ranges covering DN200 through DN500. Leakage characterization of these valves is performed according to EN 12266-1 (industrial valves, testing) and ISO 5208 (industrial valves, pressure testing), which impose specific acceptance criteria on leakage rates at defined differential pressures.

CFD simulation is used at this facility to predict valve flow coefficients (Cv), leakage rates, and pressure recovery factors across the full range of valve opening positions prior to physical testing. Each simulation campaign covers a minimum of four opening positions (typically 25%, 50%, 75%, and 100%) with additional positions added as needed for certification. Prior to the deployment of this script, each position required approximately 45 minutes of analyst time: opening the case, configuring the mesh parameters, launching the run, monitoring convergence, and progressing to the next position. Analyst attention was therefore required continuously throughout the campaign.

Post-deployment, scenario preparation time collapsed to under three minutes: the analyst edits the scenario

range in the parameter block, verifies the mesh parameters, and launches the script. The remainder of the campaign runs unattended. For a four-position campaign this represents a reduction in analyst engagement time from approximately three hours to under fifteen minutes, or approximately ninety percent.

6.2 Mesh Quality Observations

Operating with the baseline parameter set — Edge_Growth_Rate of 1.05, Volume_Growth_Rate of 1.05, Number_Of_Layers of 3, and Layer_Factor of 0.45 — the generated meshes consistently exhibit the following characteristics across all tested valve geometries: smooth cell-size transitions at the boundary layer to bulk mesh interface with no abrupt jumps in cell volume ratio; consistent non-dimensional wall distance distributions along valve seat and plug surfaces within the y+ range required for SST accuracy; no detected negative-volume cells across the full scenario range; and well-conditioned surface meshes with maximum aspect ratios below the specified Surface_Limiting_Aspect_Ratio threshold.

The Surface_Limiting_Aspect_Ratio parameter proved particularly sensitive for valve geometries. A value of 20 was found to be the practical upper bound for accurate Cv prediction: higher values introduced measurable numerical diffusion across the thin-gap seat region, manifesting as an underprediction of the flow coefficient at partial opening positions. Values below 15 produced acceptable accuracy but increased cell counts to the point where campaign run times became impractical for daily production use. The value of 20 in the baseline parameter block reflects the result of a mesh sensitivity study conducted prior to full deployment.

6.3 Convergence Behaviour

With Intelligent Solution Control enabled and the convergence thresholds as specified in the parameter block, the solver consistently identifies converged solutions within the following iteration ranges: 800 to 1200 iterations for fully open (100%) valve positions, where the flow field is dominated by a single high-velocity jet through the fully open metering gap; 1100 to 1600 iterations for the 75% opening position; 1400 to 2000 iterations for the 50% opening position, where significant recirculation begins to develop on the downstream face of the plug; and 1800 to 2800 iterations for the 25% position, where the near-seat flow field is fully three-dimensional and exhibits strong unsteady characteristics even in the time-averaged sense.

The field fluctuation threshold of 1x10^-7 enforces a strict stopping criterion that was found necessary to prevent premature convergence acceptance in the near-seat flow field. At less stringent values (10^-5 to 10^-6), the solver would occasionally satisfy the residual slope criteria while the pressure distribution on the seat surface was still evolving, producing Cv values that differed from the fully converged result by up to 3%, which is within but near the edge of the acceptable tolerance for certification use.

7. EXTENSIBILITY AND FURTHER DEVELOPMENT

The script in its current form is a single-file, single-pass tool. The architecture is intentionally minimal: a parameter block and a loop. This minimalism is a feature rather than a limitation — it makes the script easy to understand, audit, and hand over to other analysts. Several natural extensions are architecturally straightforward and follow directly from the N x 1-cycle structure.

7.1 Per-Cycle Residual Monitoring

The most immediate extension is the addition of per-cycle residual polling. After each scenario in the loop, the script could query the results object for residual history and compare the most recent residual slope against a threshold. If the threshold is exceeded — indicating divergence or stagnation — the loop could abort the current scenario, log the failure, and advance to the next index. This would prevent the common failure mode of running additional mesh cycles on a solution that has already become unphysical.

7.2 Parametric Mesh Sensitivity Studies

The parameter block can be extended to accept lists rather than scalar values for any mesh control parameter. A second outer loop over the list would then produce a full parameter sweep across scenarios, automatically covering the mesh sensitivity space. This extension is particularly useful during the initial calibration of the parameter block for a new valve family, where the optimal Surface_Limiting_Aspect_Ratio and Layer_Factor values are not yet established.

7.3 Automated Results Extraction

A results extraction block appended after each scenario loop iteration could query the Results and DC modules of the Autodesk CFD API to extract Cv, differential pressure, mass flow rate, and velocity contour data directly. These could be written to a structured CSV or Excel file without returning to the GUI. Combined with a plotting script, this would produce a complete campaign report — Cv versus opening position curve, convergence histories, mesh statistics — without any post-processing manual effort.

7.4 Transferability to Other Platforms

The broader principle demonstrated here — that decomposing a commercial software package's built-in N-cycle routine into N scripted 1-cycle calls yields disproportionate control gains — is likely transferable to other CFD platforms that expose automation interfaces. ANSYS Fluent via PyFluent, STAR-CCM+ via its Java macro layer, and OpenFOAM via its Python scripting environment are all candidates for analogous decomposition strategies, though the specific API calls and state management mechanisms will differ.

8. CONCLUSIONS

A Python automation script has been developed and deployed against the Autodesk CFD API to restructure the native adaptive meshing workflow. The central contribution is an architectural one: replacing a single native N-cycle adaptation call with N sequential scripted 1-cycle calls, separated by explicit mesh removal, parameter application, and study save operations. This transformation converts an opaque iterative process into a transparent, controllable, and auditable pipeline.

The key findings of this work can be summarized as follows. First, the N x 1-cycle decomposition is mathematically equivalent to the native N-cycle call at the level of individual refinement decisions, while offering fundamentally superior control granularity. Second, the explicit removeMesh() reset between cycles is the critical implementation detail that enables this equivalence: without it, accumulated refinement state would invalidate the decomposition. Third, the separation of operator parameters from automation logic makes the script both accessible to non-programmer engineers and suitable for version-controlled quality management.

In the industrial application at the Rho valve manufacturer, the methodology reduced analyst engagement time per simulation campaign by approximately ninety percent, eliminated parameter configuration errors arising from manual GUI interactions, and produced a version-controlled record of the mesh generation procedure for every simulation result. These gains were achieved with no reduction in mesh quality or solution accuracy relative to the manual baseline, as verified against physical leakage test data under EN 12266-1 and ISO 5208.

The methodology is presented not merely as a workflow automation convenience but as a structural improvement to the adaptation strategy itself: it elevates the mesh refinement history from an internal software state — invisible, non-recoverable, and non-auditable — to a first-class engineering artefact that is persistent, reproducible, and subject to systematic analysis.

Pedro Henrique Rodrigues de Carvalho Castello

Politecnico di Milano · Milan, Italy · 2025