Fluid Dynamics and Co-located Conferences
June 24-27, 2013, San Diego, CA
21st AIAA Computational Fluid Dynamics Conference

Adjoint-Based Error Estimation and Mesh Adaptation for the Active Flux Method

Kaihua Ding*, Krzysztof J. Fidkowski†, and Philip L. Roe‡
Department of Aerospace Engineering, University of Michigan, Ann Arbor, MI 48109

In this paper we extend output-based error estimation to recently-developed active-flux schemes. We use the active flux method for its ability to efficiently incorporate third-order accuracy for convection-dominated problems. The error estimation requires a discrete unsteady adjoint solution, for which we provide implementation details and verification demonstrations. We also introduce a localization strategy in which the adjoint-weighted residual inner product is split into contributions from various elements and nodes to create an adaptive indicator. We demonstrate the ability of these adaptative indicators to drive mesh refinement in two-dimensional advection, and we show favorable comparisons to uniform mesh refinement.

I. Introduction

The quest for accuracy in computational fluid dynamics, especially for aerospace engineering applications, has driven the development and application of high-order methods. Various high-order methods have been developed for convection-dominated flows, including high-order finite volume, streamline-upwind Petrov-Galerkin (SUPG) finite elements,[1] discontinuous Galerkin (DG) finite elements,[2] and hybridized[3] and discontinuous Petrov-Galerkin methods.[4] A recent workshop,[5] pitted some of these methods against traditional "work-horse" second-order finite volume schemes, and arrived at the conclusion that high-order approximation is beneficial for a variety of cases relevant to aerospace engineering. However, challenges remain, one of these being cost in memory and computational time.

Recently-introduced active-flux schemes[6,7] address the issue of computational cost relative to many high-order discretizations. Much like hybridized discontinuous Galerkin (HDG) methods, active flux schemes introduce additional independent variables at edges and nodes. In triangles, this doubles the degrees of freedom available to describe the solution on each cell, without enlarging the stencil. In particular, quadratic reconstructions in space and time are possible, yielding formally third-order accuracy. In contrast to HDG methods, whose efficiency relies on static condensation applied to the linear system in an implicit setting, an active flux discretization is fully discrete and can be marched forward in time using an explicit solution update strategy.

While an inexpensive high-order method is certainly desirable, such a method itself does not guarantee accuracy for a given problem. Discretization errors will still be present in the presence of non-zero mesh sizes. In this paper we therefore tackle another quest in computational fluid dynamics, that of robustness. We use the active flux method as the discretization for its ability

*Graduate Research Assistant, AIAA Member
†Assistant Professor, AIAA Senior Member
‡Professor, AIAA Fellow

to efficiently incorporate third-order accuracy, but we go further and equip the method with error estimates and adaptive capability.

The error estimation is output-based,[8] requiring the solution of an auxiliary adjoint problem. We extend previous work in output-error estimation and mesh adaptation for finite volume and finite-element schemes to the active-flux scheme. Specifically, we present a discrete-adjoint solution method, together with error estimation and localization strategies. We further use these strategies to drive mesh adaptation for the active flux schemes. Quantitative comparisons between uniform mesh refinement and our adaptive mesh refinement mechanics shows that the adjoint-driven mesh adaptation mechanics developed in this paper are more advantageous, at least for the advection problems tested.

II. Active Flux Discretization

The name of the active flux scheme is a direct reference to the fact that interface values are updated independently from conserved quantities.[9] In a traditional scheme, the flux at an interface is determined by the solution to a Riemann problem using reconstructions of conserved variables as the input. We refer to this type of update as a passive flux because the interface quantity is derived or interpolated from conserved quantities. An active flux is computed directly from edge values in a way that depends both on previous cell values and previous edge values.

In order to build the theory of the adjoint for the active flux (AF) method, we consider a scalar advection problem as the starting point,

$$\frac{\partial u}{\partial t} + \mathbf{a} \cdot \nabla u = 0. \tag{1}$$

The AF forward discretization[6,7,10] differs from traditional finite volume discretizations by defining degrees of freedom at element interfaces, which for triangles means at edge midpoints and at mesh nodes.

A. One Dimensional Active Flux Discretization

In one spatial dimension, the extra degrees of freedom are just "vertex" unknowns at the mesh nodes, as illustrated in Figure 1. These unknowns are evolved together with the cell averages

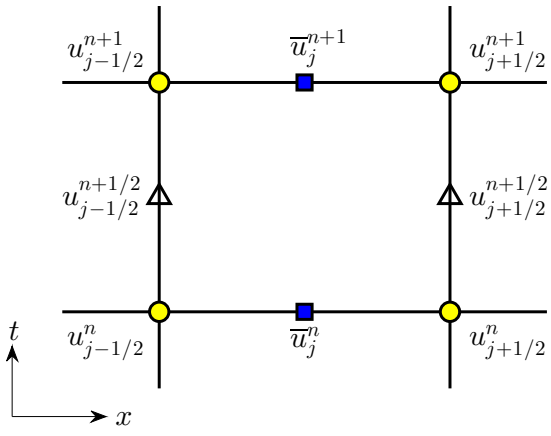


Figure 1. Unknowns placement in an AF scheme for a problem in one spatial dimension.

through the following fully-discrete procedure.

$$\text{Reform}: \begin{cases} u^{n+1/2}_{j+1/2} &= u^n_{j+1/2} \\ u^{n+1}_{j+1/2} &= u^n_{j+1/2} \end{cases} \tag{2}$$

$$\text{Vertex state pre-update:} \begin{cases} u^n_{j+1/2} &= S(\delta t = 0, c^n_j) \\ u^{n+1/2}_{j+1/2} &= S(\delta t = 0.5\Delta t, c^n_j) \\ u^{n+1}_{j+1/2} &= S(\delta t = \Delta t, c^n_j) \end{cases} \tag{3}$$

$$\text{Flux calculation: } \overline{F}^{n+1}_{j+1/2} = \frac{1}{6}(u^n_{j+1/2} + 4u^{n+1/2}_{j+1/2} + u^{n+1}_{j+1/2}) \tag{4}$$

$$\text{Update:} \begin{cases} \overline{u}^{n+1}_j &= \overline{u}^n_j - \dfrac{a\Delta t}{\Delta x}(\overline{F}^{n+1}_{j+1/2} - \overline{F}^{n+1}_{j-1/2}) \\ u^n_{j+1/2} &= u^{n+1}_{j+1/2} \end{cases} \tag{5}$$

The above update equations make use of the following definitions,

$$S(\delta t, c^n_j) = \begin{cases} RC & \xi_0 \in [0,1] \\ 0 & \xi_0 \notin [0,1] \end{cases} \tag{6}$$

$$c^n_j : \begin{cases} c^n_{j,1} &= u^n_{j-1/2} \\ c^n_{j,2} &= \dfrac{1}{4}\left(-u^n_{j-1/2} + 6\overline{u}^n_j - u^n_{j+1/2}\right) \\ c^n_{j,3} &= u^n_{j+1/2} \end{cases} \tag{7}$$

$$RC = c^n_{j,1}\left(1 - 2\xi_{0,j+1/2}\right)\left(1 - \xi_{0,j+1/2}\right) + 4c^n_{j,2}\xi_{0,j+1/2}\left(1 - \xi_{0,j+1/2}\right) + c^n_{j,3}\xi_{0,j+1/2}\left(2\xi_{0,j+1/2} - 1\right) \tag{8}$$

$$\xi_{0,j+1/2} = \underbrace{\xi}_{\text{node reference coordinate}} - a\frac{\delta t}{\Delta x} \tag{9}$$

In the AF scheme, vertex states($\mathbf{u}$) and elemental average states($\overline{\mathbf{u}}$) are defined as two independent variables. However, the update processes of $\mathbf{u}$ and $\overline{\mathbf{u}}$ are interwoven as shown above. This fact is important for the derivation of discrete adjoint formulation for the active flux method.

We can get a brief idea on the performance of the AF scheme by testing it with a suite of waveforms selected by Zalesak[11] and others. The Zalesak wave propagates across the computational domain and returns to its original position as a result of periodic boundary conditions. The solution is illustrated in Figure 2. Using our AF implementation and with one choice of element size, the shape of the wave is qualitatively similar after one period of propagation. The Zalesak waveforms qualitatively examine the performance of the AF scheme in solving square wave, cosine wave, Gaussian wave, and elliptic wave advection problems. Furthermore, we quantitatively examined the performance of the AF scheme using a smooth initial condition, a Gaussian wave. The following L_2 solution error norm, e_{L_2}, is adopted to measure the error:

$$e_{L_2} = \sqrt{\frac{1}{L}\int_0^L \left[u_{\text{AF}}(x) - u_{\text{exact}}(x)\right]^2 dx} = \sqrt{\frac{1}{L}\sum_{j=1}^{M}\left\{\int_{(j-1)\Delta x}^{j\Delta x}\left[u_{\text{AF}}(x) - u_{\text{exact}}(x)\right]^2 dx\right\}}. \tag{10}$$

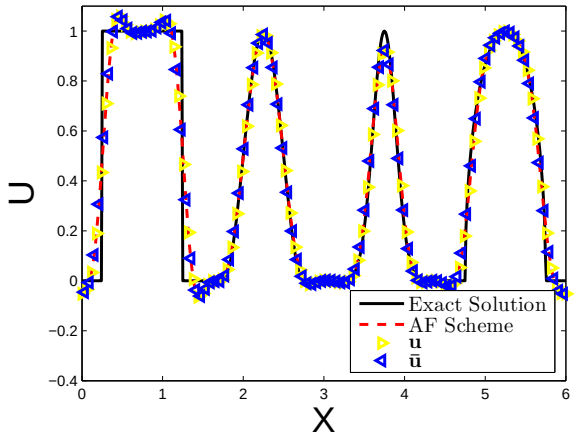
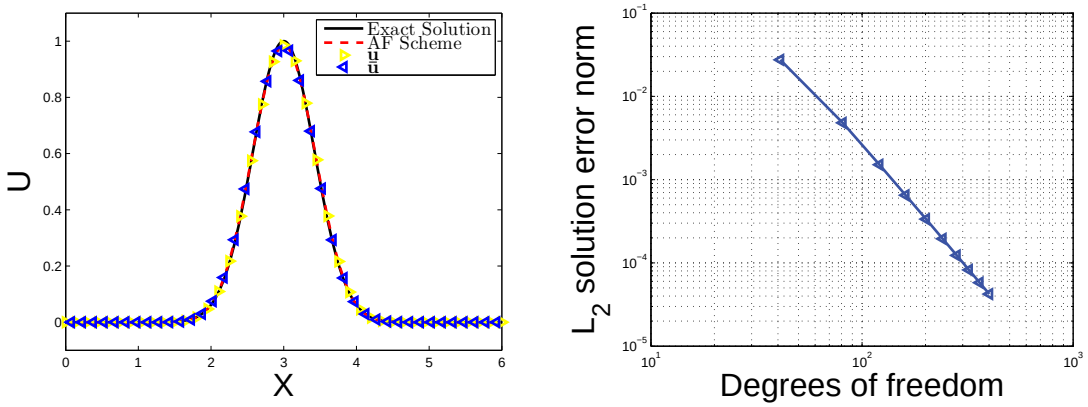


Figure 2. Zalesak wave suite propagation after one period with the AF scheme.

Eqn. 10 is evaluated with a sixth-order quadrature rule. At each quadrature point, $u_{\text{exact}}(x)$ is obtained by substituting x into the analytic Gaussian function and $u_{\text{AF}}(x)$ is obtained by using the spatial reconstruction function(Eqn. 8). Thereupon, we obtain,

$$e_{L_2} = \sqrt{\frac{1}{L}\sum_{j=1}^{M}\sum_{i=1}^{6}\left\{w_i \Delta x \left[u_{\text{AF}}(\tilde{\xi}_{j,i}) - u_{\text{exact}}(\tilde{\xi}_{j,i})\right]^2\right\}}. \tag{11}$$

The convergence rate plot of AF scheme is shown in Figure 3. Ignoring the first three data points in Figure 3(b) and applying a least squares fit to the rest of the data points with a linear function, we obtain a slope of $-2.996946490044623 \approx -3$, consistent with the third-order accuracy expectation for the AF scheme.



(a) Gaussian wave test case

(b) Convergence rate plot

Figure 3. Demonstration of third order convergence for the AF scheme in one dimension.

B. Two Dimensional Active Flux Discretization

In two spatial dimensions, the scalar advection problem takes the form,

$$\frac{\partial u}{\partial t} + a_1 \frac{\partial u}{\partial x_1} + a_2 \frac{\partial u}{\partial x_2} = 0. \tag{12}$$

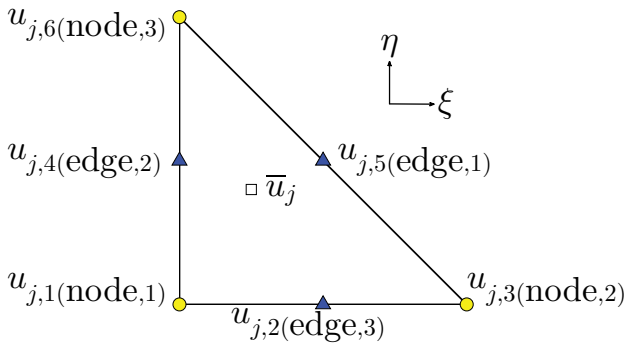


Figure 4. Unknowns placement in AF scheme in two-dimension.

Element interface values now consist of two types, node values, u_{node}, and edge values, u_{edge}. Cell average unknowns, $\overline{u}$, are defined exclusively inside of the host cells. The unknown placement in the reference triangle element is shown in Figure 4. The temporal evolution of states in two-dimensions is similar to that in one dimension.

Edge and node unknowns are defined as two types of independent variables in the active flux scheme, but they have the same evolution procedure. We can classify them into a new unknown category, vertex unknowns, u_{vertex}. This classification will simplify our discrete adjoint formulation derivation.

Two dimensional unknowns evolved through the following fully-discrete procedure.

$$\text{Reform}: \begin{cases} u^{n+1/2}_{j,1(2,3,4,5,6)} &= u^{n}_{j,1(2,3,4,5,6)} \\ u^{n+1}_{j,1(2,3,4,5,6)} &= u^{n}_{j,1(2,3,4,5,6)} \end{cases} \tag{13}$$

$$\text{Vertex state pre-update}: \begin{cases} u^{n}_{j,1(2,3,4,5,6)} &= S(\delta t = 0, c^n_j) \\ u^{n+1/2}_{j,1(2,3,4,5,6)} &= S(\delta t = 0.5\Delta t, c^n_j) \\ u^{n+1}_{j,1(2,3,4,5,6)} &= S(\delta t = \Delta t, c^n_j) \end{cases} \tag{14}$$

$$\text{Flux}: \mathbf{F}^{n+1} = \frac{1}{9}\left[\frac{1}{4}\left(\mathbf{f}^n_{\text{L}} + \mathbf{f}^n_{\text{R}} + \mathbf{f}^{n+1}_{\text{L}} + \mathbf{f}^{n+1}_{\text{R}}\right) + \left(\mathbf{f}^{n+1}_{\text{Mid}} + \mathbf{f}^{n+1/2}_{\text{L}} + \mathbf{f}^{n+1/2}_{\text{R}} + \mathbf{f}^{n}_{\text{Mid}}\right) + 4\mathbf{f}^{n+1/2}_{\text{Mid}}\right] \tag{15}$$

The subscript, 'Mid', denotes the midpoint of an element edge, which is edge state(u_2, u_4 and u_5). The subscript 'L' and 'R' denote the node state to the left and to the right of the edge state(u_1, u_3 and u_6) respectively. The fluxes are,

$$\begin{array}{lll} \mathbf{f}^n_{\text{L}} = \mathbf{a}u^n_{\text{L}} & \mathbf{f}^{n+1/2}_{\text{L}} = \mathbf{a}u^{n+1/2}_{\text{L}} & \mathbf{f}^{n+1}_{\text{L}} = \mathbf{a}u^{n+1}_{\text{L}} \\ \mathbf{f}^n_{\text{R}} = \mathbf{a}u^n_{\text{R}} & \mathbf{f}^{n+1/2}_{\text{R}} = \mathbf{a}u^{n+1/2}_{\text{R}} & \mathbf{f}^{n+1}_{\text{R}} = \mathbf{a}u^{n+1}_{\text{R}} \\ \mathbf{f}^n_{\text{Mid}} = \mathbf{a}u^n_{\text{Mid}} & \mathbf{f}^{n+1/2}_{\text{Mid}} = \mathbf{a}u^{n+1/2}_{\text{Mid}} & \mathbf{f}^{n+1}_{\text{Mid}} = \mathbf{a}u^{n+1}_{\text{Mid}} \end{array} \tag{16}$$

$$\text{Update}: \begin{cases} \overline{u}^{n+1}_j &= \overline{u}^n_j - \dfrac{\Delta t}{A_j}\displaystyle\sum_{e=1}^{3} \vec{\mathbf{n}}_e l_e \mathbf{F}^{n+1}_e \\ u^n_{j,1(2,3,4,5,6)} &= u^{n+1}_{j,1(2,3,4,5,6)} \end{cases} \tag{17}$$

The above update equations make use of the following definitions,

$$S(\delta t, c_j^n) = \begin{cases} RC & \{(\xi_0, \eta_0) \ | \xi_0 \in [0,1] \text{ and } \eta_0 \in [0, (1-\xi_0)]\} \\ 0 & \{(\xi_0, \eta_0) \ | \xi_0 \notin [0,1] \text{ or } \quad \eta_0 \notin [0, (1-\xi_0)]\} \end{cases} \tag{18}$$

$$c_j^n : \begin{cases} c_{j,1}^n &= u_{j,1}^n \\ c_{j,2}^n &= u_{j,2}^n \\ c_{j,3}^n &= u_{j,3}^n \\ c_{j,4}^n &= u_{j,4}^n \\ c_{j,5}^n &= u_{j,5}^n \\ c_{j,6}^n &= u_{j,6}^n \\ c_{j,7}^n &= \dfrac{20}{9}\left[\overline{u}_j^n - \dfrac{1}{3}\left(u_{j,2}^n + u_{j,4}^n + u_{j,5}^n\right)\right] \end{cases} \tag{19}$$

$$\phi_{j(1,2,3,4,5,6)}^n : \begin{cases} \phi_{j,1}^n &= 1 - 3\xi_0 + 2\xi_0^2 - 3\eta_0 + 4\xi_0\eta_0 + 2\eta_0^2 \\ \phi_{j,2}^n &= 4\xi_0 - 4\xi_0^2 - 4\xi_0\eta_0 \\ \phi_{j,3}^n &= -\xi_0 + 2\xi_0^2 \\ \phi_{j,4}^n &= 4\eta_0 - 4\xi_0\eta_0 - 4\eta_0^2 \\ \phi_{j,5}^n &= 4\eta_0\xi_0 \\ \phi_{j,6}^n &= -\eta_0 + 2\eta_0^2 \\ \phi_{j,7}^n &= 27\xi_0\eta_0 - 27\xi_0^2\eta_0 - 27\xi_0\eta_0^2 \end{cases} \tag{20}$$

$$RC|_{(1,2,3,4,5,6)} = c_{j,1}^n\phi_{j,1}^n + c_{j,2}^n\phi_{j,2}^n + c_{j,3}^n\phi_{j,3}^n + c_{j,4}^n\phi_{j,4}^n + c_{j,5}^n\phi_{j,5}^n + c_{j,6}^n\phi_{j,6}^n + c_{j,7}^n\phi_{j,7}^n \tag{21}$$

$$\begin{bmatrix} \xi_0 \\ \eta_0 \end{bmatrix} = \underbrace{\begin{bmatrix} \xi \\ \eta \end{bmatrix}}_{\text{node reference coordinate}} - \underbrace{\frac{\partial \mathbf{x}}{\partial \boldsymbol{\xi}}}_{\text{Jacobian}} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \Delta t \tag{22}$$

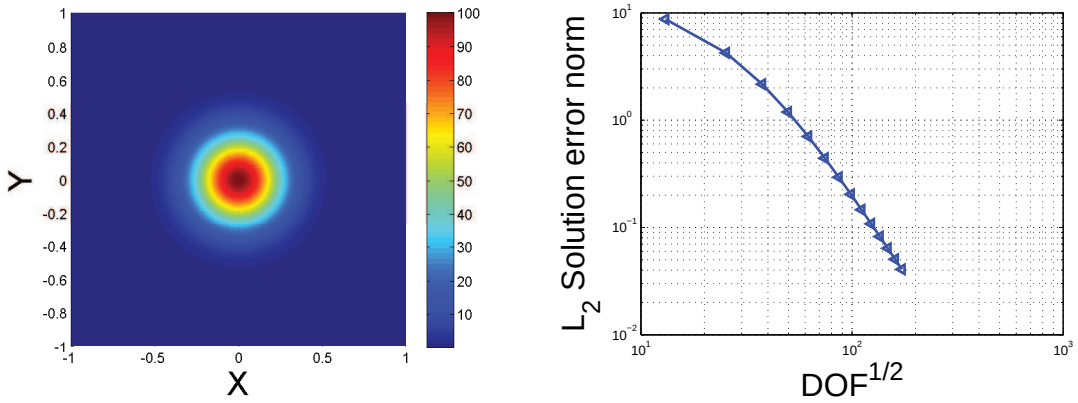
$\partial\mathbf{x}/\partial\boldsymbol{\xi}$ is the element-specific mapping Jacobian matrix. The L_2 error norm in two dimensions, e_{L_2}, is

$$e_{L_2} = \sqrt{\frac{1}{A}\int_A \left[u_{\text{AF}}(\mathbf{x}) - u_{\text{exact}}(\mathbf{x})\right]^2 dA} = \sqrt{\frac{1}{A}\sum_{j=1}^{M}\left\{\int_{\text{Element}\,j} \left[u_{\text{AF}}(\mathbf{x}) - u_{\text{exact}}(\mathbf{x})\right]^2 dA\right\}} \tag{23}$$

e_{L_2} is evaluated numerically with the sixth order quadrature rule,

$$e_{L_2} = \sqrt{\frac{1}{A}\sum_{j=1}^{M}\sum_{i=1}^{12}\left\{\left[u_{\text{AF}}(\boldsymbol{\xi}_{j,i}) - u_{\text{exact}}(\boldsymbol{\xi}_{j,i})\right]^2 \det\left(\frac{\partial \mathbf{x}}{\partial \boldsymbol{\xi}}\right)_{j,i}\right\}} \tag{24}$$

The convergence rate plot of AF scheme in 2D is shown in Figure 5(b). Figure 5(a) shows the primal problem that we run to obtain the convergence rate curve, a Gaussian pulse advecting along the diagonal of a square domain, along which periodic boundary conditions are enforced. The L_2 error norm is measured after each uniform mesh refinement. The simulation is run for one period to accentuate the effects of numerical error. The slope of the L_2 error norm convergence rate curve is gradually approaching 3 along the increase of mesh size, shown in Figure 5(b). Mesh size is represented by the square root of the total degrees of freedom in our mesh. Using the last two data points from Figure 5(b), its slope is $-2.947454128912672 \approx -3$, which is consistent with the third order accuracy expectation for the active flux method.



(a) Gaussian wave test case

(b) Convergence rate plot

Figure 5. Demonstration of third order convergence for the AF scheme in two dimensions.

III. Discrete Adjoint Formulation

A general discrete adjoint formulation of an unsteady problem reads,

$$\sum_{n=1}^{N} \left(\frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^m} \right) \mathbf{\Psi}^n + \left(\frac{\partial J}{\partial \mathbf{U}^m} \right)^T = \mathbf{0}, \tag{25}$$

where n, m index time nodes, $\mathbf{R}^n$ is the unsteady residual at time node n, $\mathbf{\Psi}^n$ is the discrete adjoint vector at time node n, and $J(\mathbf{U}^m)$ is a scalar output. Our goal is to solve the adjoint, $\mathbf{\Psi}^n$, in Eqn. 25. Eqn. 25 is a large linear system that, when fully expanded, reads

$$\begin{bmatrix} \frac{\partial \mathbf{R}^1}{\partial \mathbf{U}^1} & \frac{\partial \mathbf{R}^1}{\partial \mathbf{U}^2} & \cdots & \frac{\partial \mathbf{R}^1}{\partial \mathbf{U}^N} \\ \frac{\partial \mathbf{R}^2}{\partial \mathbf{U}^1} & \frac{\partial \mathbf{R}^2}{\partial \mathbf{U}^2} & \cdots & \frac{\partial \mathbf{R}^2}{\partial \mathbf{U}^N} \\ \vdots & & \ddots & \vdots \\ \frac{\partial \mathbf{R}^N}{\partial \mathbf{U}^1} & \frac{\partial \mathbf{R}^N}{\partial \mathbf{U}^2} & \cdots & \frac{\partial \mathbf{R}^N}{\partial \mathbf{U}^N} \end{bmatrix}^T \begin{bmatrix} \mathbf{\Psi}^1 \\ \mathbf{\Psi}^2 \\ \vdots \\ \mathbf{\Psi}^N \end{bmatrix} + \begin{bmatrix} \left(\frac{\partial J}{\partial \mathbf{U}^1} \right)^T \\ \left(\frac{\partial J}{\partial \mathbf{U}^2} \right)^T \\ \vdots \\ \left(\frac{\partial J}{\partial \mathbf{U}^N} \right)^T \end{bmatrix} = \mathbf{0} \tag{26}$$

We now need to define $\mathbf{R}$ and $\mathbf{U}$ for the AF method. In one dimension, the state vector, $\mathbf{U}$, consists of both the vertex and the cell-average unknowns,

$$\mathbf{U} = \begin{bmatrix} \mathbf{u} \\ \overline{\mathbf{u}} \end{bmatrix} \tag{27}$$

In two dimensions, to acquire a 'cleaner-looking' adjoint equation, we adopt the idea introduced in Section II, that of classifying the node unknown, $\mathbf{u}_{\text{node}}$, and edge unknown $\mathbf{u}_{\text{edge}}$ into a new category, vertex unknowns, $\mathbf{u}_{\text{vertex}}$.

$$\mathbf{U} = \begin{bmatrix} \mathbf{u}_{\text{node}} \\ \mathbf{u}_{\text{edge}} \\ \overline{\mathbf{u}} \end{bmatrix} \tag{28}$$

Moreover, we designate $\mathbf{u}_{\text{vertex}}$ via $\mathbf{u}$ in 2D. Through this way, we transform the 2D AF unknown vector, Eqn. 28 into the same format with the 1D AF unknown vector, Eqn. 27.

The residual $\mathbf{R}$, is defined in a straightforward manner as,

$$\mathbf{R}^{n+1} = \mathbf{U}^{n+1} - AF\left(\mathbf{U}^{n}\right) \tag{29}$$

where $AF(\cdot)$ is the $n \to n{+}1$ active flux update operator detailed in Section II. Eqn. (29) may also be written as

$$\begin{pmatrix} \mathbf{R}^{n+1}_{\text{vertex}} \\ \mathbf{R}^{n+1}_{\text{cell avg}} \end{pmatrix} = \begin{pmatrix} \mathbf{u}^{n+1} \\ \overline{\mathbf{u}}^{n+1} \end{pmatrix} - AF \begin{pmatrix} \mathbf{u}^{n} \\ \overline{\mathbf{u}}^{n} \end{pmatrix} \tag{30}$$

The derivative $\partial \mathbf{R}^n / \partial \mathbf{U}^m$ yields an unsteady Jacobian matrix,

$$\frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^m} = \begin{bmatrix} \dfrac{\partial \mathbf{R}^n_{\text{vertex}}}{\partial \mathbf{u}^m} & \dfrac{\partial \mathbf{R}^n_{\text{vertex}}}{\partial \overline{\mathbf{u}}^m} \\ \dfrac{\partial \mathbf{R}^n_{\text{cell avg}}}{\partial \mathbf{u}^m} & \dfrac{\partial \mathbf{R}^n_{\text{cell avg}}}{\partial \overline{\mathbf{u}}^m} \end{bmatrix} \tag{31}$$

All four block components of $\partial \mathbf{R}^n / \partial \mathbf{U}^m$ can be calculated by applying the chain rule to the AF scheme update equations.

While the general unsteady Jacobian in Eqn. 26 appears daunting, most blocks are zeros. Since the definition of $\mathbf{R}^{n+1}$ only involves states at two time steps, n and $(n+1)$, the $(n+1)^{\text{th}}$ block row of $\partial \mathbf{R}^n / \partial \mathbf{U}^m$, consists only of two non-zero terms: $\partial \mathbf{R}^{n+1} / \partial \mathbf{U}^n$ and $\partial \mathbf{R}^{n+1} / \partial \mathbf{U}^{n+1}$. Thus, the adjoint equation becomes

$$\begin{bmatrix} \dfrac{\partial \mathbf{R}^1}{\partial \mathbf{U}^1} & & & \\ \dfrac{\partial \mathbf{R}^2}{\partial \mathbf{U}^1} & \dfrac{\partial \mathbf{R}^2}{\partial \mathbf{U}^2} & & \\ & \ddots & & \\ & \dfrac{\partial \mathbf{R}^N}{\partial \mathbf{U}^{N-1}} & \dfrac{\partial \mathbf{R}^N}{\partial \mathbf{U}^N} \end{bmatrix}^T \begin{bmatrix} \boldsymbol{\Psi}^1 \\ \boldsymbol{\Psi}^2 \\ \vdots \\ \boldsymbol{\Psi}^N \end{bmatrix} + \begin{bmatrix} \left(\dfrac{\partial J}{\partial \mathbf{U}^1}\right)^T \\ \left(\dfrac{\partial J}{\partial \mathbf{U}^2}\right)^T \\ \vdots \\ \left(\dfrac{\partial J}{\partial \mathbf{U}^N}\right)^T \end{bmatrix} = \mathbf{0} \tag{32}$$

Our problem narrows down to calculating the derivatives, $\partial \mathbf{R}^{n+1} / \partial \mathbf{U}^{n+1}$ and $\partial \mathbf{R}^{n+1} / \partial \mathbf{U}^{n}$. Considering first the latter,

$$\frac{\partial \mathbf{R}^{n+1}}{\partial \mathbf{U}^n} = \begin{bmatrix} \dfrac{\partial \mathbf{R}^{n+1}_{\text{vertex}}}{\partial \mathbf{u}^n} & \dfrac{\partial \mathbf{R}^{n+1}_{\text{vertex}}}{\partial \overline{\mathbf{u}}^n} \\ \dfrac{\partial \mathbf{R}^{n+1}_{\text{cell avg}}}{\partial \mathbf{u}^n} & \dfrac{\partial \mathbf{R}^{n+1}_{\text{cell avg}}}{\partial \overline{\mathbf{u}}^n} \end{bmatrix} \tag{33}$$

The above block sub-matrices need to be calculated individually. We show how to determine $\partial \mathbf{R}^{n+1}_{\text{vertex}} / \partial \mathbf{u}^n$; other derivative matrices can be determined similarly.

A. One Dimensional Discrete Adjoint Formulation

In one spatial dimension, the expression for $\mathbf{R}^{n+1}_{\text{vertex}}$ reads,

$$\mathbf{R}^{n+1}_{\text{vertex}} = \begin{bmatrix} R_{1/2} \\ R_{1+1/2} \\ \vdots \\ R_{M+1/2} \end{bmatrix}^{n+1} = \begin{bmatrix} u_{1/2} \\ u_{1+1/2} \\ \vdots \\ u_{M+1/2} \end{bmatrix}^{n+1} - AF_{\text{vertex}}\left(\mathbf{U}^n\right) \tag{34}$$

To simplify notation, we name the vertex state update process of the AF scheme as a function: $AF_{\text{vertex}}(\cdot)$. In addition, we index the vertex residuals and states by half indices. From Eqn. 34, the derivative $\partial \mathbf{R}^{n+1}_{\text{vertex}}/\partial \mathbf{u}^n$ should be an $(M+1)\times(M+1)$ matrix,

$$\frac{\partial \mathbf{R}^{n+1}_{\text{vertex}}}{\partial \mathbf{u}^n} = \begin{bmatrix} \dfrac{\partial R^{n+1}_{1/2}}{\partial u^n_{1/2}} & \dfrac{\partial R^{n+1}_{1/2}}{\partial u^n_{1+1/2}} & \cdots & \dfrac{\partial R^{n+1}_{1/2}}{\partial u^n_{M+1/2}} \\ \dfrac{\partial R^{n+1}_{1+1/2}}{\partial u^n_{1/2}} & \dfrac{\partial R^{n+1}_{1+1/2}}{\partial u^n_{1+1/2}} & \cdots & \dfrac{\partial R^{n+1}_{1+1/2}}{\partial u^n_{M+1/2}} \\ \vdots & & \ddots & \vdots \\ \dfrac{\partial R^{n+1}_{M+1/2}}{\partial u^n_{1/2}} & \dfrac{\partial R^{n+1}_{M+1/2}}{\partial u^n_{1+1/2}} & \cdots & \dfrac{\partial R^{n+1}_{M+1/2}}{\partial u^n_{M+1/2}} \end{bmatrix} \tag{35}$$

Each row in Eqn. 35 corresponds to a derivative of $R^{n+1}_{j+1/2}$ w.r.t. $\mathbf{u}^n$. Recall the discretization of the active flux method in Section II, the $(j+1)^{\text{th}}$ row of Eqn. 35:

$$\frac{\partial R^{n+1}_{j+1/2}}{\partial \mathbf{u}^n} = \frac{\partial R^{n+1}_{j+1/2}}{\partial AF_{\text{vertex}}(\mathbf{U})^n} \frac{\partial AF_{\text{vertex}}(\mathbf{U})^n}{\partial S^n} \left[\frac{\partial S^n}{\partial c^n_{j,1}} \frac{\partial c^n_{j,1}}{\partial \mathbf{u}^n} + \frac{\partial S^n}{\partial c^n_{j,2}} \frac{\partial c^n_{j,2}}{\partial \mathbf{u}^n} + \frac{\partial S^n}{\partial c^n_{j,3}} \frac{\partial c^n_{j,3}}{\partial \mathbf{u}^n} \right] \tag{36}$$

In the above equation, $n \in [0, 1, \cdots, N]$, $j \in [0, 1, \cdots, M]$, and

$$\begin{aligned} \frac{\partial R^{n+1}_{j+1/2}}{\partial AF_{\text{vertex}}(\mathbf{U}^n)} &= -1 \\ \frac{\partial AF_{\text{vertex}}(\mathbf{U}^n)}{\partial S^n} &= 1 \\ \frac{\partial S^n}{\partial c^n_{j,1}} &= (1-2\xi_{0,j+1/2})(1-\xi_{0,j+1/2}) \\ \frac{\partial S^n}{\partial c^n_{j,2}} &= 4\xi_{0,j+1/2}(1-\xi_{0,j+1/2}) \\ \frac{\partial S^n}{\partial c^n_{j,3}} &= \xi_{0,j+1/2}(2\xi_{0,j+1/2}-1) \\ \frac{\partial c^n_{j,1}}{\partial \mathbf{u}^n} &= \left[0,\ \cdots,\ 1_{j-1/2},\ \cdots,\ 0\right] \\ \frac{\partial c^n_{j,2}}{\partial \mathbf{u}^n} &= \left[0,\ \cdots, -\frac{1}{4}_{j-1/2},\ -\frac{1}{4}_{j-1/2},\ \cdots,\ 0\right] \\ \frac{\partial c^n_{j,3}}{\partial \mathbf{u}^n} &= \left[0,\ \cdots,\ 1_{j+1/2},\ \cdots,\ 0\right] \end{aligned} \tag{37}$$

With a similar procedure, we obtain the rest of the components of the derivative $\partial \mathbf{R}^{n+1}/\partial \mathbf{U}^n$ and $\partial \mathbf{R}^{n+1}/\partial \mathbf{U}^{n+1}$. Both $\partial \mathbf{R}^{n+1}/\partial \mathbf{U}^n$ and $\partial \mathbf{R}^{n+1}/\partial \mathbf{U}^{n+1}$ turn out to be invariant in time. Moreover, $\partial \mathbf{R}^{n+1}/\partial \mathbf{U}^{n+1}$ is an identity matrix. Let $A \equiv \partial \mathbf{R}^{n+1}/\partial \mathbf{U}^n$; the unsteady discrete adjoint formulation for the active flux method then becomes

$$\begin{bmatrix} I & A^T & & \\ & I & A^T & \\ & & \ddots & \\ & & & A^T \\ & & & I \end{bmatrix} \begin{bmatrix} \boldsymbol{\Psi}^1 \\ \boldsymbol{\Psi}^2 \\ \vdots \\ \boldsymbol{\Psi}^N \end{bmatrix} + \begin{bmatrix} \left(\dfrac{\partial J}{\partial \mathbf{U}^1}\right)^T \\ \left(\dfrac{\partial J}{\partial \mathbf{U}^2}\right)^T \\ \vdots \\ \left(\dfrac{\partial J}{\partial \mathbf{U}^N}\right)^T \end{bmatrix} = \mathbf{0} \tag{38}$$

We are especially interested in the structure of the A matrix, which is sparse. For the case of an $M = 20$ element mesh, the A matrix structure is illustrated in Figure 6.

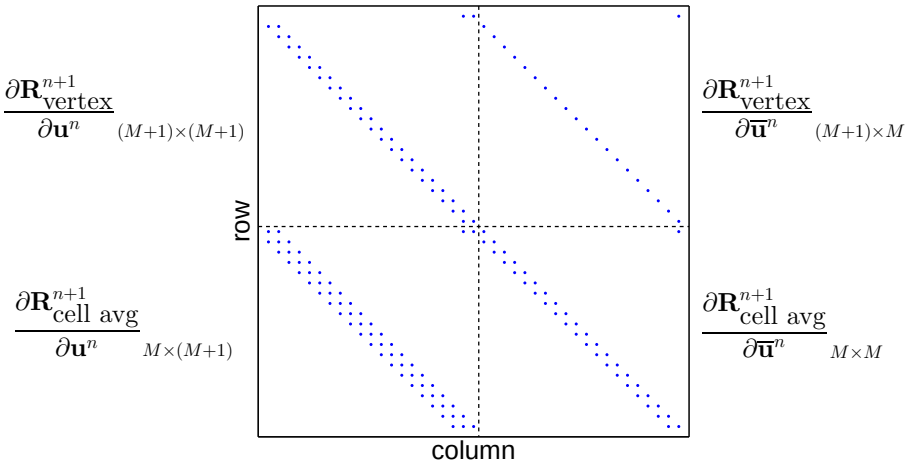


Figure 6. *A* **matrix structure in one dimension, cell number** $M = 20$ **cells.**

B. Two Dimensional Discrete Adjoint Formulation

In two spatial dimensions, assume there are M elements and K vertices, including nodes and edges, in our mesh. Here, the expression for $\mathbf{R}^{n+1}_{\text{vertex}}$ reads,

$$\mathbf{R}^{n+1}_{\text{vertex}} = \begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ R_K \end{bmatrix}^{n+1} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_K \end{bmatrix}^{n+1} - AF_{\text{vertex}}\left(\mathbf{U}^n\right) \tag{39}$$

The derivative $\partial \mathbf{R}^{n+1}_{\text{vertex}}/\partial \mathbf{u}^n$ is a $K \times K$ matrix,

$$\frac{\partial \mathbf{R}^{n+1}_{\text{vertex}}}{\partial \mathbf{u}^n} = \begin{bmatrix} \frac{\partial R_1^{n+1}}{\partial u_1^n} & \frac{\partial R_1^{n+1}}{\partial u_2^n} & \cdots & \frac{\partial R_1^{n+1}}{\partial u_K^n} \\ \frac{\partial R_2^{n+1}}{\partial u_1^n} & \frac{\partial R_2^{n+1}}{\partial u_2^n} & \cdots & \frac{\partial R_2^{n+1}}{\partial u_K^n} \\ \vdots & & \ddots & \vdots \\ \frac{\partial R_K^{n+1}}{\partial u_1^n} & \frac{\partial R_K^{n+1}}{\partial u_2^n} & \cdots & \frac{\partial R_K^{n+1}}{\partial u_K^n} \end{bmatrix} \tag{40}$$

Each row in Eqn. 40 corresponds to a derivative of R_i^{n+1} w.r.t. $\mathbf{u}^n$. Recall the discretization of the active flux method in Section II, the i^{th} row of Eqn. 40 is the residual of the i^{th} vertex state w.r.t. all other vertex states. Say, when the i^{th} vertex state is updated, the signal is from the j^{th} element. In this case, there would be at most 6 nonzero terms in the vector, $\partial {R_i}^{n+1}/\partial \mathbf{u}^n$,

$$\frac{\partial R_i^{n+1}}{\partial \mathbf{u}^n} = \begin{bmatrix} \cdots & \dfrac{\partial R_i^{n+1}}{\partial u_{j,1}^n} & \cdots & \dfrac{\partial R_i^{n+1}}{\partial u_{j,2}^n} & \cdots \dfrac{\partial R_i^{n+1}}{\partial u_{j,3}^n} & \cdots \dfrac{\partial R_i^{n+1}}{\partial u_{j,4}^n} & \cdots \dfrac{\partial R_i^{n+1}}{\partial u_{j,5}^n} & \cdots \dfrac{\partial R_i^{n+1}}{\partial u_{j,6}^n} & \cdots \end{bmatrix} \tag{41}$$

$$\frac{\partial R_i^{n+1}}{\partial \mathbf{u}^n} : \begin{cases} \dfrac{\partial R_i^{n+1}}{\partial u_{j,1}^n} &= \dfrac{\partial R_i^{n+1}}{\partial AF_{\text{vertex}}(\mathbf{U})^n} \dfrac{\partial AF_{\text{vertex}}(\mathbf{U})^n}{\partial S^n} \dfrac{\partial S^n}{\partial c_{j,1}} \dfrac{\partial c_{j,1}^n}{\partial u_{j,1}^n} \\ \dfrac{\partial R_i^{n+1}}{\partial u_{j,2}^n} &= \dfrac{\partial R_i^{n+1}}{\partial AF_{\text{vertex}}(\mathbf{U})^n} \dfrac{\partial AF_{\text{vertex}}(\mathbf{U})^n}{\partial S^n} \left(\dfrac{\partial S^n}{\partial c_{j,2}} \dfrac{\partial c_{j,2}^n}{\partial u_{j,2}^n} + \dfrac{\partial S^n}{\partial c_{j,7}} \dfrac{\partial c_{j,7}^n}{\partial u_{j,2}^n} \right) \\ \dfrac{\partial R_i^{n+1}}{\partial u_{j,3}^n} &= \dfrac{\partial R_i^{n+1}}{\partial AF_{\text{vertex}}(\mathbf{U})^n} \dfrac{\partial AF_{\text{vertex}}(\mathbf{U})^n}{\partial S^n} \dfrac{\partial S^n}{\partial c_{j,3}} \dfrac{\partial c_{j,3}^n}{\partial u_{j,3}^n} \\ \dfrac{\partial R_i^{n+1}}{\partial u_{j,4}^n} &= \dfrac{\partial R_i^{n+1}}{\partial AF_{\text{vertex}}(\mathbf{U})^n} \dfrac{\partial AF_{\text{vertex}}(\mathbf{U})^n}{\partial S^n} \left(\dfrac{\partial S^n}{\partial c_{j,4}} \dfrac{\partial c_{j,4}^n}{\partial u_{j,4}^n} + \dfrac{\partial S^n}{\partial c_{j,7}} \dfrac{\partial c_{j,7}^n}{\partial u_{j,4}^n} \right) \\ \dfrac{\partial R_i^{n+1}}{\partial u_{j,5}^n} &= \dfrac{\partial R_i^{n+1}}{\partial AF_{\text{vertex}}(\mathbf{U})^n} \dfrac{\partial AF_{\text{vertex}}(\mathbf{U})^n}{\partial S^n} \left(\dfrac{\partial S^n}{\partial c_{j,5}} \dfrac{\partial c_{j,5}^n}{\partial u_{j,5}^n} + \dfrac{\partial S^n}{\partial c_{j,7}} \dfrac{\partial c_{j,7}^n}{\partial u_{j,5}^n} \right) \\ \dfrac{\partial R_i^{n+1}}{\partial u_{j,6}^n} &= \dfrac{\partial R_i^{n+1}}{\partial AF_{\text{vertex}}(\mathbf{U})^n} \dfrac{\partial AF_{\text{vertex}}(\mathbf{U})^n}{\partial S^n} \dfrac{\partial S^n}{\partial c_{j,6}} \dfrac{\partial c_{j,6}^n}{\partial u_{j,6}^n} \end{cases} \tag{42}$$

In the above equation,

$$\begin{array}{rclrcl} \dfrac{\partial R_i^{n+1}}{\partial AF_{\text{vertex}}} &=& -1 & \dfrac{\partial c_{j,1}}{\partial u_{j,1}} &=& 1 \\ \dfrac{\partial AF_{\text{vertex}}(\mathbf{U})^n}{\partial S^n} &=& 1 & \dfrac{\partial c_{j,2}}{\partial u_{j,2}} &=& 1 \\ \dfrac{\partial S^n}{\partial c_{j,1}} &=& \phi_{j,1}^n & \dfrac{\partial c_{j,3}}{\partial u_{j,3}} &=& 1 \\ \dfrac{\partial S^n}{\partial c_{j,2}} &=& \phi_{j,2}^n & \dfrac{\partial c_{j,4}}{\partial u_{j,4}} &=& 1 \\ \dfrac{\partial S^n}{\partial c_{j,3}} &=& \phi_{j,3}^n & \dfrac{\partial c_{j,5}}{\partial u_{j,5}} &=& 1 \\ \dfrac{\partial S^n}{\partial c_{j,4}} &=& \phi_{j,4}^n & \dfrac{\partial c_{j,6}}{\partial u_{j,6}} &=& 1 \\ \dfrac{\partial S^n}{\partial c_{j,5}} &=& \phi_{j,5}^n & \dfrac{\partial c_{j,7}}{\partial u_{j,2}} &=& -\dfrac{20}{27} \\ \dfrac{\partial S^n}{\partial c_{j,6}} &=& \phi_{j,6}^n & \dfrac{\partial c_{j,7}}{\partial u_{j,4}} &=& -\dfrac{20}{27} \\ \dfrac{\partial S^n}{\partial c_{j,7}} &=& \phi_{j,7}^n & \dfrac{\partial c_{j,7}}{\partial u_{j,5}} &=& -\dfrac{20}{27} \end{array} \tag{43}$$

Following a similar procedure, we obtained the rest of the components of the derivative $\partial \mathbf{R}^{n+1}/\partial \mathbf{U}^n$ and $\partial \mathbf{R}^{n+1}/\partial \mathbf{U}^{n+1}$. Note that we again define $A \equiv \partial \mathbf{R}^{n+1}/\partial \mathbf{U}^n$. The structure of the A matrix is shown in Figure 7. The mesh that we used to generate this particular A matrix is illustrated in Figure 15(a). Obviously, one dimensional A matrix and two dimensional A matrix are different. Since the unsteady discrete adjoint formulation for the active flux method in two dimensions has the same formulation with its one dimensional counterpart, by subsituting different A matrix into Eqn. 38, we will be able to get the adjoint solution in both 1D and 2D.

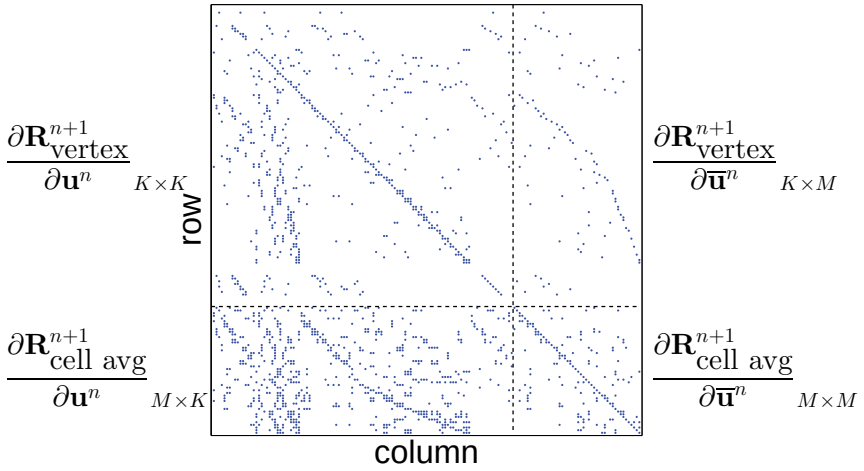


Figure 7. *A* **matrix structure in two dimensions, cell number** $M = 48$**, vertices number** $K = 113$**.**

IV. Implementation

To solve Eqn. 38, we note that the coefficient matrix in Eqn. 38 is upper triangular, and hence, backward substitution may be applied,

$$\begin{aligned} \boldsymbol{\Psi}^N &= -\left(\frac{\partial J}{\partial \mathbf{U}^N}\right)^T \\ \boldsymbol{\Psi}^{N-1} + A^T\boldsymbol{\Psi}^N &= -\left(\frac{\partial J}{\partial \mathbf{U}^{N-1}}\right)^T \\ &\vdots \\ \boldsymbol{\Psi}^1 + A^T\boldsymbol{\Psi}^2 &= -\left(\frac{\partial J}{\partial \mathbf{U}^1}\right)^T \end{aligned} \tag{44}$$

Accordingly, our adjoint code employs "reverse" time marching. The adjoint solution does not depend on the primal solution for our linear problem, yet the adjoint solve is still performed after the primal solve in anticipation of nonlinear problems.

V. Error Estimation and Adaptation

Our objective is to use the adjoint solution in order to estimate the numerical error in a scalar output, $J_H(\mathbf{U}_H)$, where the subscript H indicates a "coarse" discretization – that is, one for which we want to estimate the error. We estimate the error in J_H relative to a finer discretization, subscript h, by the adjoint-weighted residual method,[8]

$$\delta J \equiv J(\mathbf{U}_H) - J(\mathbf{U}_h) \approx -\boldsymbol{\Psi}_h^T \mathbf{R}\left(\mathbf{U}_h^H\right) \tag{45}$$

In this work, we obtain the fine space (h) by subdividing both the spatial cells and the time steps of the coarse (H) discretization. That is, in one spatial dimension, the fine mesh has $M_h = 2M_H$ cells and in two spatial dimensions, the fine mesh has $M_h = 4M_H$. On the other hand, the time steps are always halved, so that $N_h = 2N_H$. We assess the accuracy of the error estimate through

the definition of an error effectivity,

$$\eta_h = \frac{-\mathbf{\Psi}_h^T \mathbf{R}(\mathbf{U}_h^H)}{J_H(\mathbf{U}_H) - J_h(\mathbf{U}_h)}, \tag{46}$$

where $\mathbf{U}_h^H$ indicates the coarse solution injected into the fine space. This injection is performed by evaluating the quadratic spatial and temporal reconstructions implied by the AF discretization. For spatial reconstruction, we can use the existing active flux basis functions. For temporal reconstruction, we first evaluate the coefficients of the temporal reconstruction, assuming the intermediate state($u_j^{n+1/4}$) lies on a quadratic curve connecting states from adjacent time steps(u_j^n and u_j^{n+1}),

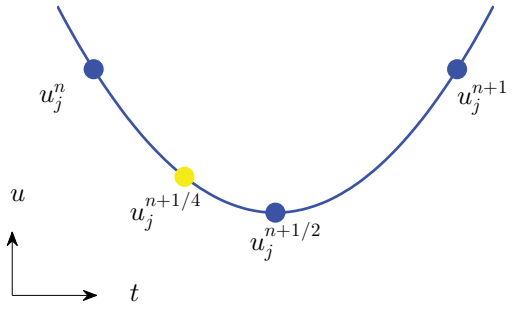


Figure 8. Temporal states injection operation mechanics.

The subscript on η_h in Eqn. 46 indicates that this is an effectivity measured relative to the fine space. We will denote the effectivity relative to the exact solution by η, and this is calculated by replacing $J_h(\mathbf{U}_h)$ in Eqn. 46 with the exact output.

Finally, in order to adapt, we need to localize the output error in Eqn. 45 to cells and to time steps. To do this we must decide how to map the residual, $\mathbf{R}(\mathbf{U}_h^H)$, back to coarse space both spatially and temporally. Regarding time nodes, we associate coarse time step $n_H, 1 \leq n_H \leq N_H$, to the residuals of the two fine-space updates contained within n_H, namely $n_h = 2n_H - 1$ and $n_h = 2n_H$. Regarding the spatial cells, we associate cell-average residuals directly to their host cells, and we split vertex residuals evenly between the adjacent cells. In one dimension, the resulting error contribution of cell j at time step n is

$$\varepsilon_j^n = \underbrace{\frac{1}{2}\mathbf{\Psi}^n_{\text{vertex},j}\mathbf{R}^n_{\text{vertex},j} + \frac{1}{2}\mathbf{\Psi}^n_{\text{vertex},j+1}\mathbf{R}^n_{\text{vertex},j+1}}_{\text{vertex residual contribution}} + \underbrace{\mathbf{\Psi}^n_{\text{cell avg},j}\mathbf{R}^n_{\text{cell avg},j}}_{\text{cell average residual contribution}} \tag{47}$$

In two dimensions, each node in our mesh is shared by 'sn' number of elements and each edge is shared by 'se' number of elements, the resulting error contribution of cell j at time step n is

$$\varepsilon_j^n = \underbrace{\sum_{i=1}^{3} \frac{1}{\text{sn}}\mathbf{\Psi}^n_{\text{node},i}\mathbf{R}^n_{\text{node},i}}_{\text{node residual contribution}} + \underbrace{\sum_{e=1}^{3} \frac{1}{\text{se}}\mathbf{\Psi}^n_{\text{edge},e}\mathbf{R}^n_{\text{edge},e}}_{\text{edge residual contribution}} + \underbrace{\mathbf{\Psi}^n_{\text{cell avg},j}\mathbf{R}^n_{\text{cell avg},j}}_{\text{cell average residual contribution}} \tag{48}$$

The error indicator for a coarse cell/time-step is then taken as the absolute value of the sum of ε_j^n over the children cells/time-steps. In particular, we will initially work with aggregate spatial

error indicators obtained by summing over all time steps, in which the error associated with coarse cell j_H is

$$\epsilon_{j_H} = \sum_{n_h=1}^{N_h} \sum_{j_h=1}^{M_h} |\varepsilon_{j_h}^{n_h}|. \tag{49}$$

VI. Results

As a verification test of the adjoint implementation, we consider the one dimensional periodic transport of the initial profile shown in Figure 9. This linear "hat" profile was chosen for the sake of error estimation: it is exactly representable on both the coarse and the fine spaces, so that the error estimates are not polluted by varying initial conditions (which, for example, would occur for any initial condition that was not at most a piecewise quadratic).

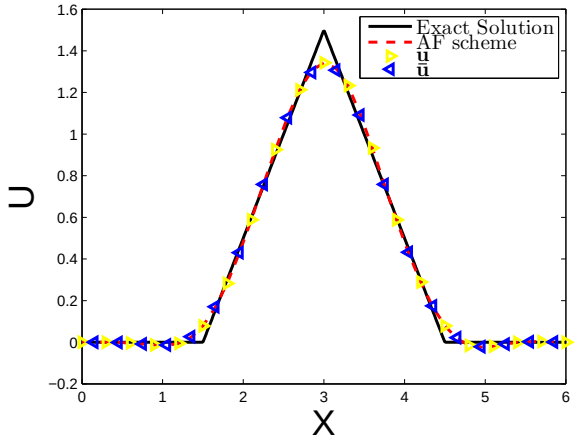
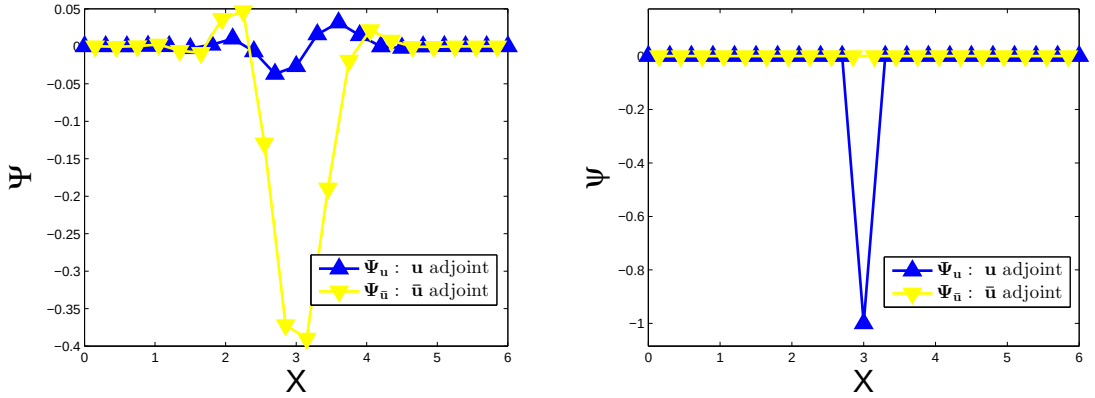


Figure 9. Linear hat wave function propagated for one period.



(a) Adjoint distribution at the first time step

(b) Adjoint distribution at the final time step

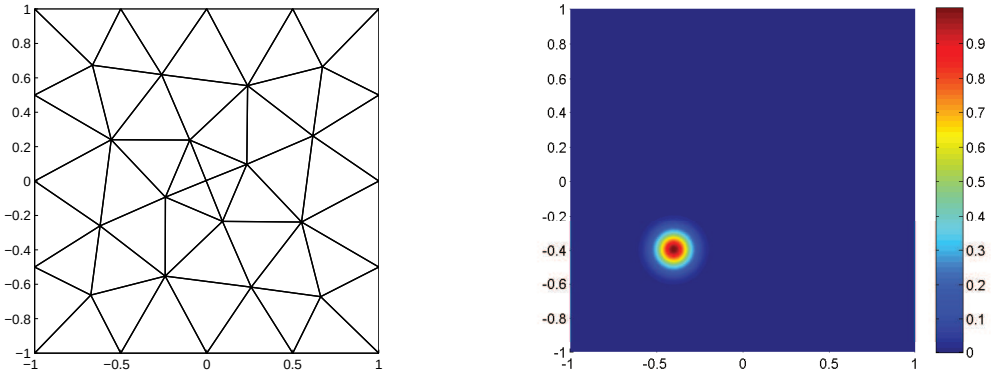
Figure 10. Adjoint distribution on our computational domain

The output J is defined as the eleventh node value ($X = 3$) at the final time step. We first take a qualitative look at the adjoint solution. Figure 10 shows the adjoint at the first time step,

i.e. $\mathbf{\Psi}^1$, and the adjoint at the final step, i.e. $\mathbf{\Psi}^N$.

The region of nonzero adjoint is wider at the initial time compared to the final time. In fact, at the final time, there is only one nonzero point, as expected since no other states can impact the output. At the initial time step, a residual perturbation in any point on the computational domain has a better chance of affecting the output as the state advects across the computational domain. So, the output is sensitive to a relatively larger area at the initial time step.

In two dimensions, we consider the test case of an advecting Gaussian wave as shown in Figure 11. Figure 11(a) shows the initial unstructured mesh, and Figure 11(b) shows the primal solution. Here, we enforced inflow boundary conditions on the left and lower boundaries of the computational domain. A Gaussian pulse originally centered at coordinate, $X = [-0.4,\ -0.4]$ advects along the diagonal of the square until it arrives the ending center point $X = [0.4,\ 0.4]$.



(a) Initial mesh

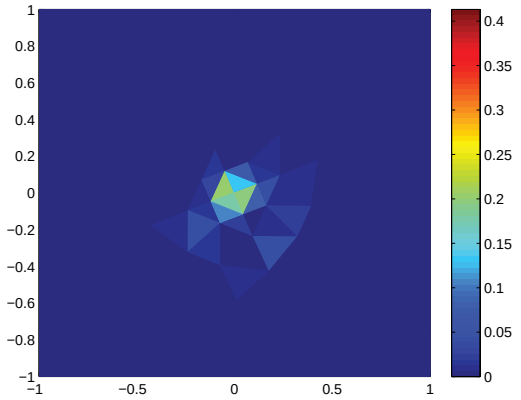
(b) Primal problem illustration

Figure 11. Initial mesh and primal solution for a two-dimensional advection problem.

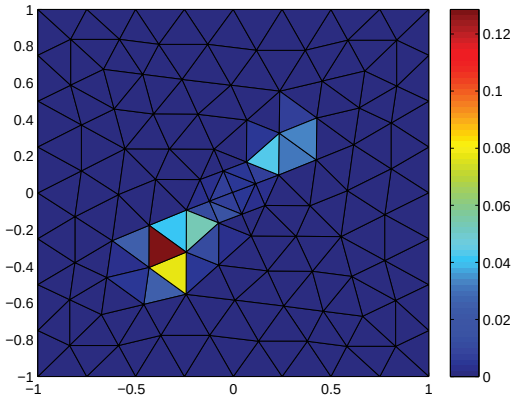
Figure 12 presents adjoints for three outputs: a point value, a domain integral, and a line integral. These plots are generated using a mesh that is the immediate fine space of the mesh shown in Figure 11(a).

The point output is defined at spatial location $[x = 0.4, y = 0.4]$. The domain integral output is defined as the state integral over the whole computational domain. The line integral output is the integral over upper and right boundaries. Recall the definition of the adjoint as the sensitivity of an output to residual source perturbations. For the point output, the 'sensitive area' has a higher adjoint value, mainly concentrated around the diagonal of our computational domain as shown in Figure 12(a). For the domain integral output, its 'sensitive area' is the area swept over by the convective flow, which is in a rectangle shape for our case as is shown in Figure 12(c). For the line integral output, its 'sensitive area' is in a corner shape, which eventually develops into the corner formed by the outflow boundary, upper and right boundaries, of our computational domain, as is shown in Figure 12(e). Figure 12(a), (c) and (e) demonstrate that our adjoint implementation is correct in a qualitative sense.

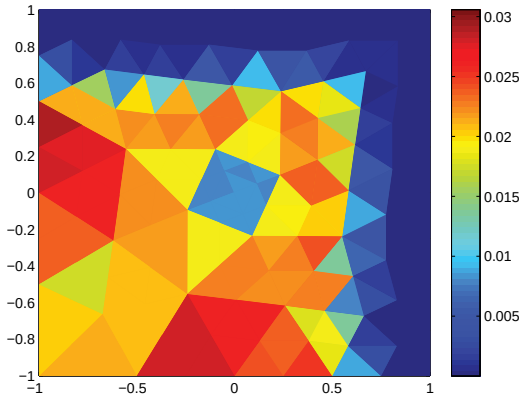
For a quantitative verification of the adjoint, we consider sensitivity analysis and error estimation.



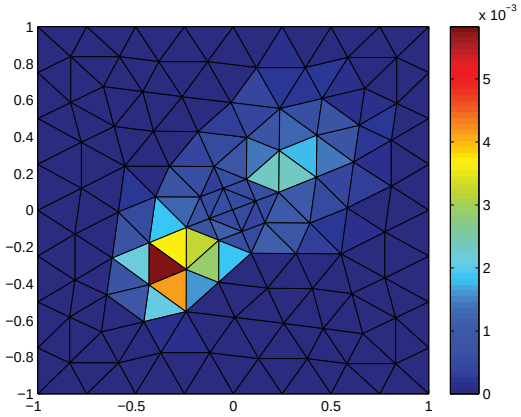
(a) Adjoint distribution for the point value



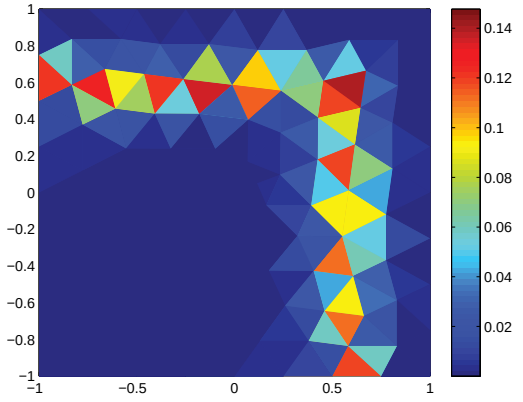
(b) Error indicator(adjoint-weighted residual) for the point value



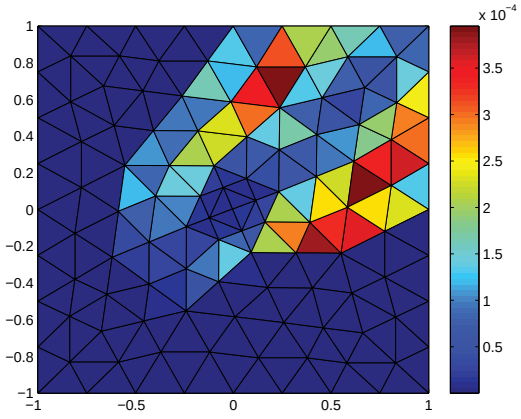
(c) Adjoint distribution for the domain integral



(d) Error indicator(adjoint-weighted residual) for the domain integral



(e) Adjoint distribution for the line integral



(f) Error indicator(adjoint-weighted residual) for the line integral

Figure 12. 2D adjoint and error indicator distribution at time step $N_h/2$

A. Sensitivity Analysis

Given an unsteady adjoint solution, a parameter sensitivity is calculated as

$$\frac{dJ}{d\mu} = \sum_{n=1}^{N} \mathbf{\Psi}^n \left(\frac{\partial \mathbf{R}^n}{\partial \mu} \right) \tag{50}$$

In this case, the parameter μ is chosen to be the amplitude of a spatially-sinusoidal perturbation to the initial condition. Theoretically, with the sensitivity information, we can predict the change of the output J along a change in the parameter, $\delta\mu$, without running the forward solver. This output perturbation is given by

$$\delta J = \left(\frac{dJ}{d\mu} \right) \delta\mu \tag{51}$$

We choose to perturb state, $\mathbf{U}$, to measure the accuracy of our adjoint cases. The calculation of the output derivative, $dJ/d\mathbf{U}$, is done utilizing the quadratic spatial formulation implied by the active flux method. For the one-dimensional case, the derivative calculation is simple. For the two-dimensional case, we briefly show how to calculate derivatives of three types of output: point output, line integral output and domain integral output. Quantitative verification of sensitivity tests are also shown in this paper.

1. $dJ/d\mathbf{U}$ *for Point Output*

In order to calculate derivative for point output, we need to first locate the element j, that contains the desired point output. Recall the active flux discretization from Section II,

$$\frac{dJ}{d\mathbf{U}} = \Bigg[\underbrace{\cdots \phi_{j,1} \cdots \phi_{j,3} \cdots \phi_{j,6} \cdots}_{\text{node states}} \quad \underbrace{\cdots \frac{27\phi_{j,2} - 20\phi_{j,7}}{27} \cdots \frac{27\phi_{j,4} - 20\phi_{j,7}}{27} \cdots \frac{27\phi_{j,5} - 20\phi_{j,7}}{27} \cdots}_{\text{edge states}} \quad \underbrace{\cdots \frac{20\phi_{j,7}}{9} \cdots}_{\text{cell average states}} \Bigg] \tag{52}$$

2. $dJ/d\mathbf{U}$ *for Line Integral Output*

Our line integral output is defined as the integral over the outflow boundary,

$$J_{\text{line integral}} = \sum_{j=1}^{\text{BCline}} J_j \tag{53}$$

J_j is line integral at the boundary edge of element j, which is located on the outflow boundary. We uses 1D quadrature to integrate J_j numerically. Thus,

$$\frac{dJ_{\text{line integral}}}{d\mathbf{U}} = \sum_{j=1}^{\text{BCline}} \frac{dJ_j}{d\mathbf{U}} \tag{54}$$

and

$$\frac{dJ_j}{d\mathbf{U}} = \sum_{q=1}^{Q} \det \left(\frac{\partial \mathbf{x}}{\partial \boldsymbol{\xi}} \right)_q \frac{dJ_q}{d\mathbf{U}} w_q \tag{55}$$

w_q is the weight of 1D quadrature point evaluated at point output J_q, Q is the total number of quadrature points.

3. $dJ/d\mathbf{U}$ for Domain Integral Output

When the output is defined as the integral over the computational domain,

$$J_{\text{domain integral}} = \sum_{j=1}^{M} J_j \tag{56}$$

J_j is area integral of the state over element j. We uses 2D quadrature rule to calculate it numerically.

$$\frac{dJ_{\text{domain integral}}}{d\mathbf{U}} = \sum_{j=1}^{M} \frac{dJ_j}{d\mathbf{U}} \tag{57}$$

and

$$\frac{dJ_j}{d\mathbf{U}} = \sum_{q=1}^{Q} \det\left(\frac{\partial \mathbf{x}}{\partial \boldsymbol{\xi}}\right)_q \frac{dJ_q}{d\mathbf{U}} w_q \tag{58}$$

w_q is the weight of the q^{th} 2D quadrature point, Q is the total number of quadrature points.

4. Sensitivity Test

To test the adjoint-based sensitivity, we run the active flux solver twice: first without a parameter perturbation, i.e. $\mu = 0$, and second with a parameter perturbation, $\mu = \delta\mu$. The output perturbation should be predicted with the adjoint sensitivity. A comparison of the actual perturbation and the predicted perturbation for the test case under consideration is given in Table 1

Table 1. Initial-condition sensitivity test comparing an actual output perturbation with an adjoint-based sensitivity calculation. For the linear problem and output under consideration, the adjoint result is exact.

Output types	Actual perturbation	Predicted perturbation
1D point output	0.039062093143630	0.039062093143630
2D point output	-11.828244710966711	-11.828244710966711
2D line integral	2.243744802369696	2.243744802369697
2D domain integral	-21.666898164171524	-21.666898164171496

As Table 1 shows, the output perturbation is predicted exactly. We can draw two conclusions here: (1) our theory for the active-flux adjoint implementation is working; (2) because the current problem is linear, the prediction is exact, but this will not be the case for general nonlinear problems.

B. Error Estimation

We now apply the adjoint to estimate the error in the output relative to a finer discretization, using the adjoint-weighted residual in Eqn. 45. The error is compared with actually solving the problem on the finer discretization. The comparison is presented in Table 2. As Table 2 shows, the error estimate is accurate up to machine precision. The associated effectivity is $\eta_h \approx 1$, also up to machine precision.

Table 2. Error estimation test

Output types	Actual error	Estimated error
1D point output	-0.062837526306201	-0.062837526306203
2D point output	-0.168407731651310	-0.168407731651310
2D line integral	-0.023801197849757	-0.023801197849757
2D domain integral	$3.226858860222309 \times 10^{-4}$	$3.226858860222372 \times 10^{-4}$

For the one-dimensional case, we further verified the convergence rate of the absolute value of the estimated error($|\epsilon_{jH}|$) and the actual error($|J_H(\mathbf{U}_H) - J_h(\mathbf{U}_h)|$) on a smooth problem, advection of a Gaussian wave. Figure 13 presents 10 pairs of data points for $|\epsilon_{jH}|$ and $|J_H(\mathbf{U}_H) - J_h(\mathbf{U}_h)|$. Applying a least square fit to the data points in Figure 13, and ignoring the first 4 data points, the slope of the $|J_H(\mathbf{U}_H) - J_h(\mathbf{U}_h)|$ data set is $-2.970697749024125 \approx -3$ and the slope of the $|\epsilon_{jH}|$ data set is $-2.96935936216698 \approx -3$, which is again consistent with the expectation of third-order accuracy for AF schemes. We note that the error estimate for an arbitrary initial condition will generally not be accurate up to machine precision according to our discussion at the beginning of Section VI (we need exact representation on the coarse space).

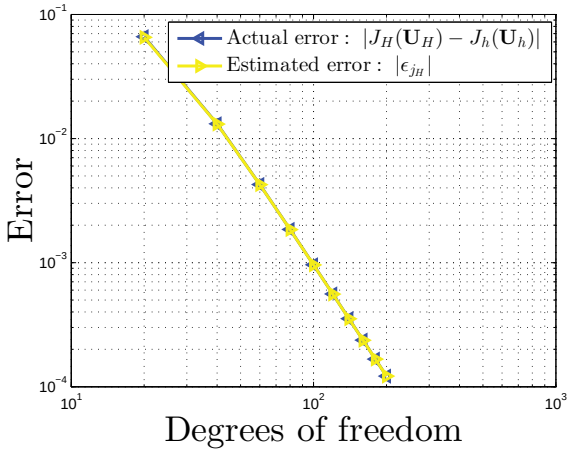


Figure 13. Convergence rate of $|J_H(\mathbf{U}_H) - J_h(\mathbf{U}_h)|$ and $|\epsilon_{jH}|$ for a Gaussian wave advection problem.

We next consider the error indicator for adaptivity by plotting the temporally-marginalized (summed over time steps) error indicator, ϵ_{jH} from Eqn. 47, versus cell number in Figure 14. From Figure 14 we see that the output error indicator is fairly evenly distributed over the domain, with slightly larger values near the output measurement location due to the high sensitivity of that area near later times. The uniformity of the error indicator for this simple problem is expected as in a periodic wave propagation on a static mesh, the entire domain requires resolution. Adaptive refinement is therefore not necessary in such a case, but we expect it to be important for more complex, higher-dimensional problems, i.e, two dimensions.

We present the error indicators of the two-dimensional cases in Figure 12(b), (d) and (f), which are calculated with the error localization strategy in Eqn. 48. Comparing the left hand side of Figure 12(Figure 12 (a), (c), (e)), which quantify the sensitivity information, adjoint, and the right hand side of Figure 12(Figure 12(b), (d), (f)), which quantify the localized discretization error information, adjoint weighted residual, we roughly have an idea about how different the sensitivity and localized error estimation information is.

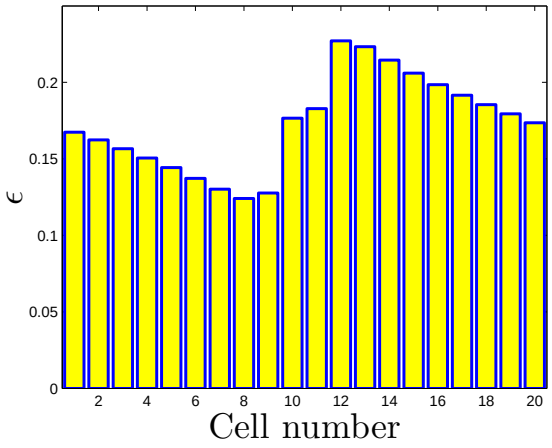


Figure 14. Temporally-marginalized adaptive indicator, ϵ_{jH} from Eqn. 47.

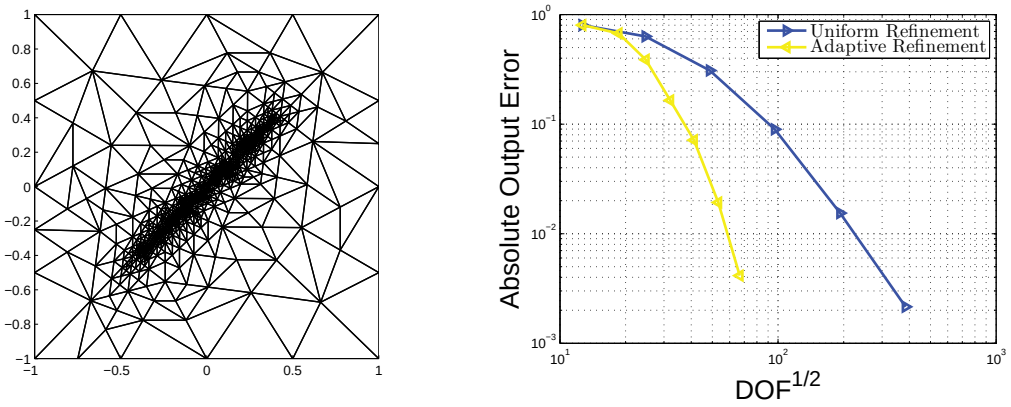
We noticed, in Figure 12(b), the point output error indicator distribution is rather uneven. Few elements have high error indicator values, but most elements have almost zero error indicator value. In terms of this particular output, it means some parts of our mesh have especially high discretization error and some parts of the mesh don't have much discretization error. In other words, some parts of the mesh need to be refined to reduce the discretization error, while some parts of the mesh don't need to be refined at all. We can expect adaptive mesh refinement to be more advantageous over uniform mesh refinement in this case. Thus, we evaluate the performance of our theoretical work using a point output. As for the line integral over the outflow boundary and domain integral output, their values basically depend on the information of the whole computational domain or almost the whole computational domain. It's not necessary to use adaptive techniques for these two cases.

C. Mesh Adaptation

To get a better CFD simulation result, the computational mesh needs to be refined. We do not expect uniform refinement to be the optimal option. In this paper, we create an output based mesh adaptation strategy for the active flux method. To measure how good our adaptation is, we compare to uniform mesh refinement for a point output.

Using the initial mesh and primal problem we have already described in detail in Figure 11, we implemented both adaptive mesh refinement and uniform mesh refinement methods to solve this problem. The absolute output error of each adapted mesh was recorded. Comparison results are shown in Figure 15(b). Adaptive refinement utilizes cheaper machine storage and generates more accurate outputs. What's more, the difference at "degrees of freedom" cost between adaptive refinement and uniform refinement grows larger as mesh size gets bigger. Our mesh adaptation mechanics beats uniform mesh refinement. We also included the resultant adapted mesh in Figure 15(a). The diagonal parts of the mesh looks really fine, while, the region away from the diagonal of the square geometry, seems 'untouched' by the mesh adaptation mechanics.

This result makes perfect sense. Due to the convection nature of the unsteady flow field, first we understand it's not enough to only adapt the area where our output is defined. So, the mesh would not only be refined around the output point. Second, our point output ultimately arises from the upstream of the computational domain. Along the advection path of the flow, error can be introduced into the simulation, which eventually pollutes the point output. Thus, besides adapting



(a) Resulting adapted mesh

(b) Error convergence comparison between adaptative mesh refinement and uniform mesh refinment

Figure 15. Application of theoretical work on a square domain

the area where our output is defined, the area where the flow swept over during the simulation should also be refined. Third, for a localized output, point output, not all the area that the flow field swept over matters, since the point output only depends on the information from a small part of the computational domain. Figure 15(a) shows that the area around the diagonal of the computational domain has more weight in affecting the accuracy of the output.

With the purpose of illustrating our research approach, we adopted a simple square geometry in the above discussion. However, for two-dimensional cases, the computational domain can be of arbitrary shape. Using inflow and outflow boundary condition and applying our theoretical work to some interesting meshes, we further examine our theoretical work.

D. Additional Simulations

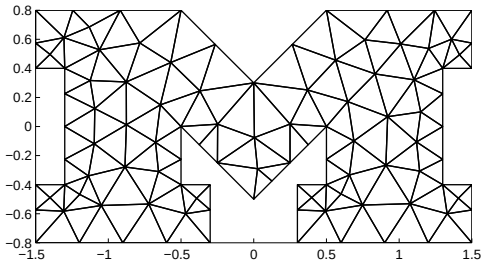
Three different computational domain geometries are presented in this section: a Michigan 'M' mesh, a circle mesh, and a "moon" mesh. These domains demonstrate to some extent the flexibility of our adaptation mechanics. We briefly describe the problem setup for each of them and present the initial mesh, the resulting adapted mesh, the primal problem and error convergence performance comparison between our mesh adaptation mechanics and uniform mesh refinement.

1. Michigan 'M' Mesh

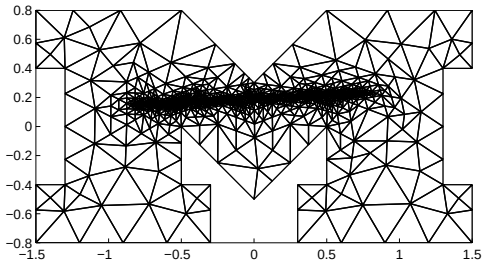
The Michigan 'M' mesh is representative of a complex polygon. Inflow and outflow boundary conditions are dynamically enforced on the border of the 'M' mesh depending on the flow field advection direction. The initial mesh is in Figure 16(a). The flow advection velocity is $[2,\ 0.1]$. A Gaussian pulse originally centered at $[-0.8,\ 0.15]$, Figure 16(c), advects with the flow until it arrives at $[0.8,\ 0.23]$. Then, the point output is measured at $[0.8,\ 0.23]$. Figure 16(d) shows adaptive mesh refinement is more desirable over uniform mesh refinement in this case.

The reason why we choose to dynamically set up the boundary condition is because, upon refinement, a curved inflow segment might be broken into an inflow segment and an outflow segment, or, the other way around, which require us to dynamically re-set the boundary condition according to the flow field convection direction. We can't see this issue for this polygon mesh, but we will benefit from this extra step for other meshes. If the boundary condition is not dynamically re-set, an inflow(outflow) boundary might be falsely recognized as a outflow(inflow) boundary after

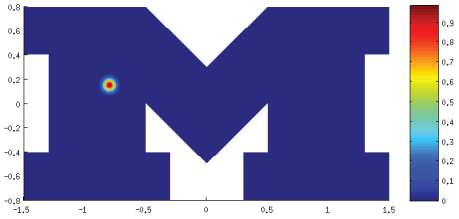
mesh adaptation. What's more, due to the uniqueness of the updating procedure of the active flux method, besides inflow/outflow 'edge' boundary condition, there also exist inflow/outflow 'node' boundary condition. That's the second reason why we need to dynamically reset the boundary condition.



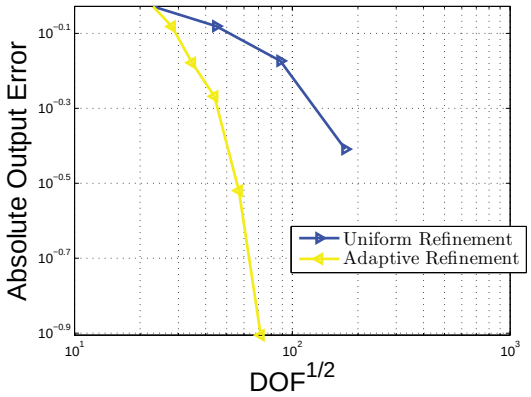
(a) Initial mesh



(b) Resulting adapted mesh



(c) Primal problem illustration

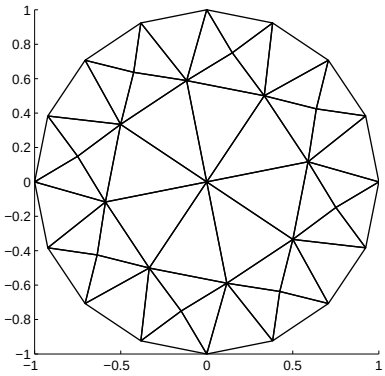


(d) Error convergence comparison between adaptative mesh refinement and uniform mesh refinment

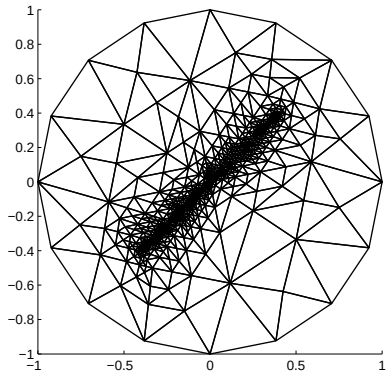
Figure 16. Application of theoretical work on Michigan 'M' mesh

2. Circular Mesh

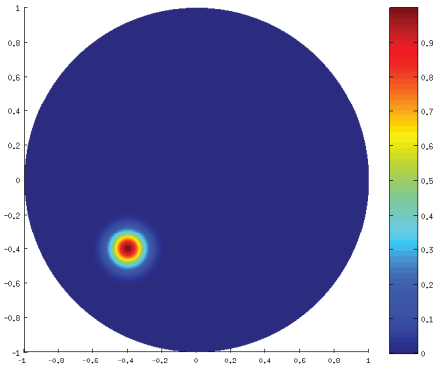
The purpose of doing this circular mesh case is to show that, besides polygon, the solver that we developed in this paper is capable of taking in geometries with curved boundary. Inflow/outflow boundary conditions are dynamically enforced on the border of the circle depending on the flow field advection direction. The initial mesh is shown in Figure 17(a). As for the set up for the primal problem and termination condition of simulation is exactly the same with the square geometry case that we described at the beginning of Section VI. The primal problem description is presented in Figure 17(c). Upon refinement, we obtain the results shown in Figure 17(d), which demonstrates that our adaptive mesh refinement method outperforms the uniform mesh refinement method.



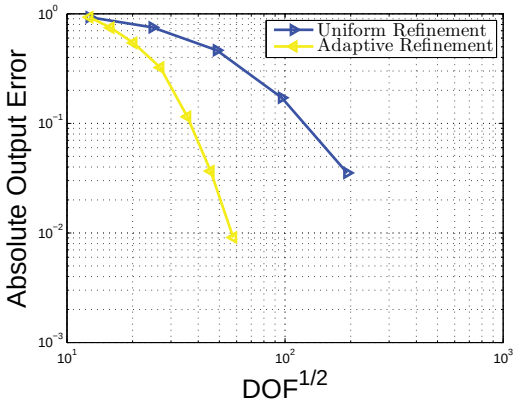
(a) Initial mesh



(b) Resulting adapted mesh



(c) Primal problem illustration



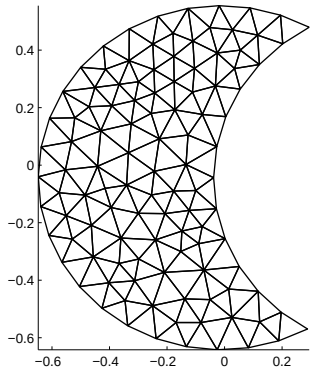
(d) Error convergence comparison between adaptative mesh refinement and uniform mesh refinment

Figure 17. Application of theoretical work on circular mesh

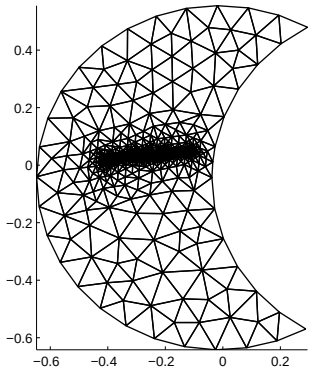
3. *Moon Mesh*

The Moon mesh is a representative of complex curved geometry. Inflow/outflow boundary conditions are dynamically enforced on the border of this mesh. The initial mesh is in Figure 18(a). Flow field advection velocity is $[1,\ 0.1]$. A Gaussian pulse originally centers at $[-0.4,\ 0]$, Figure 16(c), advects with the flow until it arrives at its ending center $[-0.1,\ 0.03]$. We apply uniform mesh refinement and adaptive mesh refinement on this problem respectively. Upon refinement, the performance of uniform mesh refinement and adaptive mesh refinement is compared in Figure 18(d). Figure 18(d) shows our adaptive mesh refinement method converge error faster than the uniform mesh refinement method.

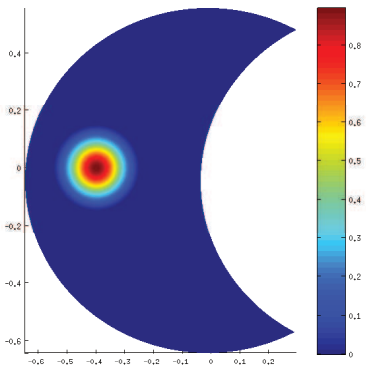
Implementing our theoretical work to various geometries enable us to further analyze our theoretical work. We can draw two conclusions here: (1)our theoretical work applies to any mesh geometries. Varying the geometries doesn't affect the performance of the developed mesh adaptation mechanics; (2)by simply observing the adapted mesh, one already have a pretty good idea about what happened in the unsteady simulation. The darkened area of mesh in Figure 15(b), Figure 16(b), Figure 17(b) and Figure 18(b) indicate where the point output that we measured advect from. The convection nature of the physics, unsteady convecting flow field, is clearly disclosed by the resulting adapted mesh.



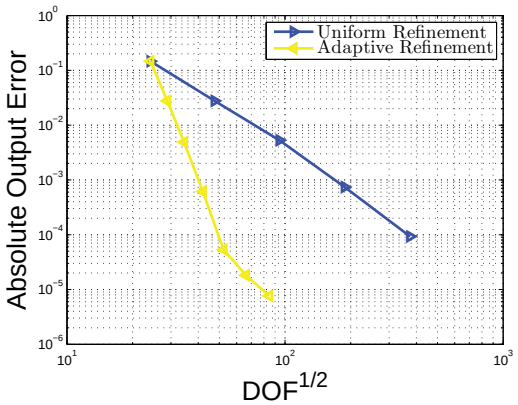
(a) Initial mesh



(b) Resulting adapted mesh



(c) Primal problem illustration



(d) Error convergence comparison between adaptative mesh refinement and uniform mesh refinment

Figure 18. Application of theoretical work on Moon mesh

VII. Conclusions

In this work, we explored the capability of the newly developed active flux methods with periodic boundary condition and inflow/outflow boundary condition, on various geometries. There are mainly two contributions in this paper,

- Equip the active flux method with error estimation capability

  The adjoint system was derived for the adtive flux method. Theoretical work was tested on linear problems, both the sensitivity test and the error estimation test passed at machine precision level. In other words, for linear problems, the theory is able to predict residual perturbations and discretization errors exactly. In general, this will not be the case for nonlinear problems.

- Create a mesh adaptation mechanics for the active flux method

  An error localization strategy is introduced in this paper, with which we created an output-based adaptive mesh refinement mechanics. The comparison between uniform mesh refinement and our adaptive mesh refinement mechanics shows, the mesh adaptation mechanics developed in this paper is more advantageous.

VIII. Acknowledgements

This work was supported by NASA, grant number NNX12AJ70A.

References

[1]Venkatakrishnan, V., Allmaras, S. R., Kamenetskii, D. S., and Johnson, F. T., "Higher order schemes for the compressible Navier-Stokes equations," AIAA Paper 2003-3987, 2003.

[2]Cockburn, B. and Shu, C.-W., "Runge-Kutta discontinuous Galerkin methods for convection-dominated problems," *Journal of Scientific Computing*, Vol. 16, No. 3, 2001, pp. 173–261.

[3]Nguyen, N., Peraire, J., and Cockburn, B., "An implicit high-order hybridizable discontinuous galerkin method for linear convection-diffusion equations," *Journal of Computational Physics*, Vol. 228, 2009, pp. 3232–3254.

[4]Demkowicz, L. and Gopalakrishnan, J., "A class of discontinuous Petrov-Galerkin methods. Part I: The transport equation," *Computer Methods in Applied Mechanics and Engineering*, Vol. 199, No. 23-24, 2010, pp. 1558–1572.

[5]Wang, Z., Fidkowski, K. J., Abgrall, R., Bassi, F., Caraeni, D., Cary, A., Deconinck, H., , Hartmann, R., Hillewaert, K., Huynh, H., Kroll, N., May, G., Persson, P.-O., van Leer, B., and Visbal, M., "High-order cfd methods: Current status and perspective," *International Journal for Numerical Methods in Fluids*, 2012, Submitted.

[6]Eymann, T. A. and Roe, P. L., "Active flux schemes," 49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition 2011–382, 2011.

[7]Eymann, T. A. and Roe, P. L., "Active flux schemes for systems," 20th AIAA Computational Fluid Dynamics Conference 2011–3840, 2011.

[8]Fidkowski, K. J. and Darmofal, D. L., "Review of output-based error estimation and mesh adaptation in computational fluid dynamics," *American Institute of Aeronautics and Astronautics Journal*, Vol. 49, No. 4, 2011, pp. 673–694.

[9]Eymann, T. A., *Active Flux Schemes*, Ph.D. thesis, The University of Michigan, Ann Arbor, 2013.

[10]van Leer, B., "Towards the ultimate conservative difference scheme iv. a new approach to numerical convection," *Journal of Computational Physics*, Vol. 23, 1977, pp. 276–299.

[11]Zalesak, S. T., "Fully multidimensional flux-corrected transport algorithms for fluids," *Journal of Computational Physics*, Vol. 31, 1979, pp. 335–362.