Efficient Computation of Jacobian

Jared Crean

January 23, 2018

1 Face Integral Overview

This section describes the high-level overview of how to differentiate the face integrals. This document assumes the interpolation from the volume nodes to the face nodes requires data from all volume nodes.

1.1 Notation and Definitions

This section describes notation that will be used throughout the document.

Julia-like notation will be used for array indexing, ie, $\boldsymbol{A}[:, j]$ returns the jth column of $\boldsymbol{A}$. This document describes the operations applied to a single face. The term "face nodes" implicitly refers to the nodes of the face under consideration.

- **numNodesPerElement**: the number of volume nodes in each element
- **numNodesPerFace**: the number of face cubature nodes
- **numDofPerNode**: the number of degrees of freedom on each node, for example, 4 for the 2D Euler equations.
- $\boldsymbol{q}_L$: a **numDofPerNode** x **numNodesPerElement** array containing the solution at the volume nodes of the left element of the interface
- $\boldsymbol{q}_R$: array of same size as $\boldsymbol{q}_L$, but holds the solution at the volume nodes of the right element of the interface
- $\boldsymbol{q}_{face,L}$: array of size **numDofPerNode** x **numNodesPerFace**, holds $\boldsymbol{q}_L$ interpolated to the face
- $\boldsymbol{q}_{face,R}$: array of same size as $\boldsymbol{q}_{face,L}$, but holds $\boldsymbol{q}_R$ interpolated to the face
- $\boldsymbol{f}$: array of size **numDofPerNode** x **numNodesPerFace**, holds the flux at each face node

- $\boldsymbol{R}_L$: array, same size as $\boldsymbol{q}_L$, holds the residual at the volume nodes of the left element
- $\boldsymbol{R}_R$: array, same size as $\boldsymbol{q}_R$, holds the residual at the volume nodes of the right element.
- $\dot{\boldsymbol{f}}_L$: array of size **numDofPerNode** x **numDofPerNode** x **numNodesPerFace**, where $\dot{\boldsymbol{f}}_L[:,:,j]$ holds the derivative of the flux with respect to the solution at the face at node j, ie. $\frac{\partial \boldsymbol{f}[:,j]}{\partial \boldsymbol{q}_{face,L}[:,j]}$.
- $\dot{\boldsymbol{f}}_R$: similar to $\dot{\boldsymbol{f}}_L$, but holds the derivative with respect to $\boldsymbol{q}_{face,R}$.
- $\dot{\boldsymbol{f}}_{volume,L}$: array of size **numDofPerNode** x **numDofPerNode** x **numNodesPerFace** x **numNodesPerElement**, holds the Jacobian of the flux at each face node with respect to the solution at each volume node of the left element, ie. $\dot{\boldsymbol{f}}_{volume,L}[:,:,j,i] = \frac{\partial \boldsymbol{f}[:,j]}{\partial \boldsymbol{q}_L[:,i]}$
- $\dot{\boldsymbol{f}}_{volume,R}$: like $\dot{\boldsymbol{f}}_{volume,L}$, but stores Jacobian with respect to $\boldsymbol{q}_R$.
- $\dot{\boldsymbol{R}}_{A,B}$:array of size **numDofPerNode** x **numDofPerNode** x **numNodesPerElement** x **numNodesPerElement**, where A and B take values of either L or R. This array holds the derivative of the residual on element A with respect to the solution on the volume nodes of element B. For example, $\dot{\boldsymbol{R}}_{L,R}[i,j,p,q] = \frac{\partial \boldsymbol{R}_L[i,p]}{\partial \boldsymbol{q}_R[j,q]}$
- $\boldsymbol{I}_L$: is an array of size **numNodesPerFace** x **numNodesPerElement** that interpolates from the volume nodes to the face nodes of the left element. This matrix is usually called $\boldsymbol{R}$ in the SBP package.
- $\boldsymbol{I}_R$: similar array to $\boldsymbol{I}_L$, but interpolates from the volume nodes of the right element to the face nodes.

1.2 Integral Computation

A pseudo-code representation of the boundary integral procedure is

---

Input: $\boldsymbol{q}_L$, $\boldsymbol{q}_R$
Data: $\boldsymbol{q}_{face,L}$, $\boldsymbol{q}_{face,R}$, $\boldsymbol{f}$
Output: $\boldsymbol{R}_L$, $\boldsymbol{R}_R$

```
# Interpolate to face:
q_face,L, q_face,R = faceInterpolate(q_L, q_R)

# Compute flux at each face node:
for j = 1:numNodesPerFace do
    f[:,j] = RoeSolver(q_L[:,j], q_R[:,j])
end

# Integrate and apply test function
R_L, R_R = faceIntegrate(f)
```

1.3 Integral Differentiation

Data: $\boldsymbol{q}_L$, $\boldsymbol{q}_R$, $\dot{\boldsymbol{f}}_L$, $\dot{\boldsymbol{f}}_R$
Interpolate to face:
$\boldsymbol{q}_{face,L}$, $\boldsymbol{q}_{face,R}$ = faceInterpolate($\boldsymbol{q}_L$, $\boldsymbol{q}_R$)

# Compute flux Jacobians at each face node
for j = *1:**numNodesPerFace*** **do**
  $\dot{\boldsymbol{f}}_L[:,:,j]$, $\dot{\boldsymbol{f}}_R[:,:,j]$ = RoeSolver_diff($\boldsymbol{q}_{face,L}[:,j]$, $\boldsymbol{q}_{face,R}[:,j]$)
end

See Section 2
$\dot{\boldsymbol{f}}_{volume,L}$ = FluxExpansion($\dot{\boldsymbol{f}}_L$)
$\dot{\boldsymbol{f}}_{volume,R}$ = FluxExpansion($\dot{\boldsymbol{f}}_R$)

See Section 3
$\dot{\boldsymbol{R}}_{L,L}$, $\dot{\boldsymbol{R}}_{R,L}$ = ReverseInterpolation($\dot{\boldsymbol{f}}_{volume,L}$)
$\dot{\boldsymbol{R}}_{R,L}$, $\dot{\boldsymbol{R}}_{R,R}$ = ReverseInterpolation($\dot{\boldsymbol{f}}_{volume,R}$)

The function RoeSolver_diff() computes the derivative of the flux with respect to $\boldsymbol{q}_{face,L}$ and $\boldsymbol{q}_{face,R}$. For reasons that will become clearer in Section 3, the ReverseInterpolation() function computes the Jacobian of the residual of both the left and right elements with respect to both its inputs at the same time.

2 Flux Expansion

The purpose of the function FluxExpansion() is to compute

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}_L} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}_{face,L}} \frac{\partial \boldsymbol{q}_{face,L}}{\partial \boldsymbol{q}_L}. \tag{1}$$

for $\boldsymbol{q}_L$ and the corresponding quantity for $\boldsymbol{q}_R$. However, because of the ordering of the data in the arrays, the operation is not a simple matrix-matrix multiplication.

2.1 Scalar Case

Considering the special case when **numDofPerNode** = 1, $\boldsymbol{q}_L$ and $\boldsymbol{q}_R$ become vectors, and the action of the $\boldsymbol{I}_L$ can be written

$$\boldsymbol{q}_{face,L} = \boldsymbol{I}_L \boldsymbol{q}_L \tag{2}$$

and therefore

$$\frac{\partial \boldsymbol{q}_{face,L}}{\partial \boldsymbol{q}_L} = \boldsymbol{I}_L. \tag{3}$$

Eq. (1) can be evaluated as

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}_L} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}_{face,L}} \boldsymbol{I}_L \tag{4}$$

where RoeSolver_diff() computes $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}_{face,L}}$.

2.2 Vector Case

In the case **numDofPerNode** > 1, the $\boldsymbol{I}_L$ operator should be applied to the **last** dimension of $\boldsymbol{f}$.

$$\boldsymbol{q}_{face,L}[j,:] = \boldsymbol{I}_L \boldsymbol{q}_L[j,:] \quad \text{for } j = 1 : \textbf{numDofPerNode}. \tag{5}$$

Correspondingly,

$$\frac{\partial \boldsymbol{f}[i,:]}{\partial \boldsymbol{q}_L[j,:]} = \frac{\partial \boldsymbol{f}[i,:]}{\partial \boldsymbol{q}_{face,L}[j,:]} \boldsymbol{I}_L \quad \text{for } i,j = 1 : \textbf{numDofPerNode}. \tag{6}$$

In cases where the flux function is applied pointwise to the face nodes, $\frac{\partial \boldsymbol{f}[i,p]}{\partial \boldsymbol{q}_{face,L}[j,q]} = 0$ when $p \neq q$, thus the matrix is diagonal. The signature of the RoeSolver() in Algorithm 1 indicates this is the case, and we will use this simplification for the remainder of the document.

Using the notation defined in Section 1.1, this can be written in code as

```
for k=1:numNodesPerElement
  for p=1:numNodesPerFace
    for i=1:numDofPerNode
      for j=1:numDofPerNode
        f_dot_volume,L[i,j,p,k] = f_dot_L[i,j,p] * I_L[p,k]
      end
    end
  end
end
```

Notice that the i and j loops are scaling $\dot{\boldsymbol{f}}_L[i,j,p]$ by the same entry of $\boldsymbol{I}_L$, which is a very efficient code pattern.

3 Reverse Interpolation

3.1 Scalar Case

Examining the case when **numDofPerNode** $= 1$, the operation that needs to be done now is to relate the derivative of the flux with respect to $\boldsymbol{q}_L$ and $\boldsymbol{q}_R$ to the derivative of the

residual. Examining the residual of the left element with respect to $\boldsymbol{q}_R$

$$\frac{\partial \boldsymbol{R}_L}{\partial \boldsymbol{q}_R} = \frac{\partial \boldsymbol{R}_L}{\partial \boldsymbol{f}} \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}_R}, \tag{7}$$

which becomes

$$\dot{\boldsymbol{R}}_{L,R} = \boldsymbol{I}_L^T \boldsymbol{B} \dot{\boldsymbol{f}}_{volume,R} \tag{8}$$

where $\boldsymbol{B}$ is the diagonal matrix of integration weights at the face nodes.

3.2 Vector Case

When **numDofPerNode** > 1, the operation becomes

$$\dot{\boldsymbol{R}}_{L,R}[i,j,:,:] = \boldsymbol{I}_L^T \boldsymbol{B} \dot{\boldsymbol{f}}_{volume,R}[i,j,:,:] \quad \text{for } i,j = 1 : \textbf{numDofPerNode} \tag{9}$$

Taking advantage of the fact that $\boldsymbol{B}$ is diagonal, this can be rewritten as

$$\dot{\boldsymbol{R}}_{L,R}[i,j,p,q] = \sum_{k=1}^{\textbf{numNodesPerFace}} \boldsymbol{I}_L^T[p,k] \boldsymbol{B}[k,k] \dot{\boldsymbol{f}}_{volume,R}[i,j,k,q] \quad \text{for } i,j = 1 : \textbf{numDofPerNode}, \quad \text{for } p,q = 1 : \textbf{numNodesPerElement} \tag{10}$$

This can be rewritten as

```
for q=1:numNodesPerElement
  for p=1:numNodesPerElement
    for k=1:numNodesPerFace
      for i=1:numDofPerNode
        for j=1:numDofPerNode
          R_dot_LR[i,j,p,q] += I_L[k,p] * B[k,k] * f_dot_volume,R[i,j,k,q]
        end
      end
    end
  end
end
```

Similar operations can be defined for $\dot{\boldsymbol{R}}_{R,R}$, $\dot{\boldsymbol{R}}_{L,L}$, and $\dot{\boldsymbol{R}}_{R,L}$. The calculation of $\dot{\boldsymbol{R}}_{L,R}$ and $\dot{\boldsymbol{R}}_{R,R}$ use the same $\dot{\boldsymbol{f}}_{volume,R}$ array and should probably be combined.

4 Combined Operations

Substituting the definition of $\dot{\boldsymbol{f}}_{volume,R}$ based on (6) into (10), it is possible to combine the two operations into one:

```
for q=1:numNodesPerElement
  for p=1:numNodesPerElement
    for k=1:numNodesPerFace
      for i=1:numDofPerNode
        for j=1:numDofPerNode
          Ṙ_{L,R}[i,j,p,q] += I_L[k,p] * B[k,k] * ḟ_R[i,j,k] * I_R[p,q]
        end
      end
    end
  end
end
```

5 Volume Terms

Compared to the face terms, the volume terms are much simpler. Some additional notation is required

5.1 Notation and Definitions

- **dim**: the number of dimensions (usually 2 or 3).
- $\boldsymbol{q}$: array of size **numDofPerNode** x **numNodesPerElement**, holds the solution at the volume nodes of the element.
- $\boldsymbol{R}$: array of size **numDofPerNode** x **numNodesPerElement**, holds the residual at each volume node.
- $\dot{\boldsymbol{R}}$: array of size **numDofPerNode** x **numDofPerNode** x **numNodesPerElement** x **numNodesPerElement**, holds the derivative of the residual with respect to the solution at the volume nodes, ie. $\dot{\boldsymbol{R}}[i,j,p,q] = \frac{\partial \boldsymbol{R}[i,p]}{\partial \boldsymbol{q}[j,q]}$
- $\boldsymbol{Q}_i$: the SBP weak differentiation operator in direction i, array of size **numNodesPerElement** x **numNodesPerElement**
- $\boldsymbol{g}$: an array of size **numDofPerNode** x **numNodesPerElement** x **dim**, holds the flux at each volume node.
- $\dot{\boldsymbol{g}}$: an array of size **numDofPerNode** x **numDofPerNode** x **numNodesPerElement** x **dim**, holds the derivative of the flux with respect to the solution at the volume nodes, ie. $\dot{\boldsymbol{g}}[i,j,p,q] = \frac{\partial \boldsymbol{g}[i,p,d]}{\partial \boldsymbol{q}[j,q]}$

5.2 Scalar Case

When **numDofPerNode** $= 1$, the first dimension of $\boldsymbol{g}$ disappears and the volume term becomes

$$\boldsymbol{R} = \sum_{d}^{\mathbf{dim}} \boldsymbol{Q}_d^T \boldsymbol{g}[:, d] \tag{11}$$

The derivative can be written

$$\frac{\partial \boldsymbol{R}[p]}{\partial \boldsymbol{q}[q]} = \sum_{d}^{\mathbf{dim}} \sum_{k}^{\mathbf{numNodesPerElement}} \boldsymbol{Q}_d^T[p, k] \frac{\partial \boldsymbol{g}[k, d]}{\partial \boldsymbol{q}[q]} \tag{12}$$

Because the flux function is applied pointwise, $\frac{\partial \boldsymbol{g}[k,d]}{\partial \boldsymbol{q}[q]} = 0$ when $k \neq q$, this can be rewritten as

$$\frac{\partial \boldsymbol{R}[p]}{\partial \boldsymbol{q}[q]} = \sum_{d}^{\mathbf{dim}} \boldsymbol{Q}_d^T[p, q] \frac{\partial \boldsymbol{g}[q, d]}{\partial \boldsymbol{q}[q]}. \tag{13}$$

5.3 Vector Case

In the case when **numDofPerNode** > 1, the volume term becomes

$$\boldsymbol{R}[i, :] = \sum_{d}^{\mathbf{dim}} \boldsymbol{Q}_d^T \boldsymbol{g}[i, :, d] \quad \text{for } i = 1 : \mathbf{numDofPerNode} \tag{14}$$

and the derivative can be written

$$\frac{\partial \boldsymbol{R}[i, p]}{\partial \boldsymbol{q}[j, q]} = \sum_{d}^{\mathbf{dim}} \boldsymbol{Q}_d^T[p, q] \frac{\partial \boldsymbol{g}[i, q, d]}{\partial \boldsymbol{q}[j, q]}. \tag{15}$$

This can be written in code as

```
for d=1:dim
  for p=1:numNodesPerElement
    for q=1:numNodesPerElement
      for j=1:numDofPerNode
        for i=1:numDofPerNode
          Ṙ[i,j,p,q] += Q_d[q,p] * ġ[i,j,q,d]
        end
      end
    end
  end
end
```

6 Euler Homotopy Volume Terms

The volume terms for the homotopy are more involved than the volume terms in the previous section. Differentiating them will require some additional notation and a somewhat different set of intermediate arrays.

6.1 Notation and Definitions

This section will use the same notation as the previous section, with a few additions:

- $\boldsymbol{D}_i$: the SBP differentiation matrix in the i-th parametric direction, **numNodesPerElement** x **numNodesPerElement**
- λ_p^d: the maximum eigenvalue of the Euler flux Jacobian at node k of the element in direction d
- δ_{ij}: the Dirac delta function, with a value of 1 if i and j are equal and zero otherwise
- $t1$: a **numDofPerNode** x **numNodesPerElement** array. Values defined by Algorithm 3.
- $\dot{t1}$: a **numDofPerNode** x **numDofPerNode** x **numNodesPerElement** x **numNodesPerElement** array holding the derivative of $t1$ with respect to $\boldsymbol{q}$, ie. $\dot{t1}[i,j,p,q] = \frac{\partial t1[i,p]}{\partial q[j,q]}$
- $t2$: array, same size as $t1$. Values defined by Algorithm 3.
- $\dot{t2}$: derivative of $t2$ with respect to $\boldsymbol{q}$. Same size and layout as $\dot{t1}$.

6.2 Vector Case

The volume terms are calculated as:

Data: $\boldsymbol{q}$, $\boldsymbol{R}$
for d_1 = *1:**dim*** **do**
  $t1$[i, :] = $\boldsymbol{D}_{d_1}\boldsymbol{q}[i,:]$ for i=1:**numDofPerNode**
  $t2$[:, p] = $t1$[:, p] $\lambda_p^{d_1}$
  $\boldsymbol{R}[i,:]$ += $\boldsymbol{Q}_{d_1}^T$ $t2$[i, :] for i=1:**numDofPerNode**
end

Using explicit indices for all arrays, this can be rewritten as:

Data: $\boldsymbol{q}$, $\boldsymbol{R}$

1 **for** d_1 = *1:**dim*** **do**

2 $\quad t1$[i, p] = $\sum_c^{\textbf{numDofPerNode}} \boldsymbol{D}_i[p,c]\boldsymbol{q}[i,c]$ for i=1:**numDofPerNode**

3 $\quad t2$[i, p] = $t1$[i, p] $\lambda_p^{d_1}$ for i=1:**numDofPerNode**

4 $\quad \boldsymbol{R}[i,p]$ += $\sum_c^{\textbf{numNodesPerElement}} \boldsymbol{Q}_{d_2}^T[p,c]$ $t2$[i, c] for i=1:**numDofPerNode**

5 **end**

Differentiating line-by-line gives:

Data: $\boldsymbol{q}$, $\dot{\boldsymbol{R}}$

1 **for** d_1 = *1:**dim*** **do**

2 $\quad \dot{t1}[i,j,p,q] = \boldsymbol{D}_i[p,c]\frac{\partial \boldsymbol{q}[i,c]}{\partial \boldsymbol{q}[j,q]} = \boldsymbol{D}_{d_1}\delta_{ij}\delta_{cq}$

3 $\quad \dot{t2}[i,j,p,q] = t1[i,p]\frac{\partial \lambda_p^{d_1}}{\partial \boldsymbol{q}[j,q]} + \lambda_p^{d_1}\dot{t1}[i,j,p,q]$

4 $\quad \dot{\boldsymbol{R}}[i,j,p,q] = \sum_c^{\textbf{numNodesPerElement}} \boldsymbol{Q}_{d_1}^T[p,c]\dot{t2}[i,j,c,q]$

5 **end**

Line 2 can be further simplified as $\dot{t1}[:,:,p,q] = \boldsymbol{D}_{d_1}[p,q]$, however, its only use is in line 3. Instead, $\boldsymbol{D}_{d_1}[p,q]$ can be used directly, and $\dot{t1}$ becomes unnecessary. Additionally, line 3 can be simplified by noting $\frac{\partial \lambda_p^{d_1}}{\partial \boldsymbol{q}[j,q]} = 0$ when $p \neq q$.