High Order Reverse Mode of AD
Theory and Implementation

Mu Wang and Alex Pothen

Department of Computer Science
Purdue University

September 30, 2016
Research Overview

- Second order reverse mode:
  - More efficient in evaluating Hessian in both complexity and memory usage in many applications.
  - Proved to be equivalent to an variance of vertex elimination on the computational graph of the gradient.

- High order reverse mode:
  - High order reverse mode: evaluating derivative tensor $\nabla^d f$ up to any order in reverse mode.
  - Implementation: ReverseAD.

- Applications:
  - Uncertainty quantification.
  - Chemistry: exchange-correlation (XC) energy functional.
Research Overview

- **Second order reverse mode**: 
  - More efficient in evaluating Hessian in both complexity and memory usage in many applications.
  - Proved to be equivalent to an variance of vertex elimination on the computational graph of the gradient

- **High order reverse mode**: 
  - High order reverse mode: evaluating derivative tensor $\nabla^d f$ up to any order in reverse mode
  - Implementation: ReverseAD

- **Applications**: 
  - Uncertainty quantification
  - Chemistry: exchange-correlation (XC) energy functional
Research Overview

▶ Second order reverse mode:
  ▶ More efficient in evaluating Hessian in both complexity and memory usage in many applications.
  ▶ Proved to be equivalent to an variance of vertex elimination on the computational graph of the gradient

▶ High order reverse mode:
  ▶ High order reverse mode: evaluating derivative tensor $\nabla^d f$ up to any order in reverse mode
  ▶ Implementation: ReverseAD

▶ Applications:
  ▶ Uncertainty quantification
  ▶ Chemistry: exchange-correlation (XC) energy functional
Research Overview

- Second order reverse mode:
  - More efficient in evaluating Hessian in both complexity and memory usage in many applications.
  - Proved to be equivalent to an variance of vertex elimination on the computational graph of the gradient.

- High order reverse mode:
  - High order reverse mode: evaluating derivative tensor $\nabla^d f$ up to any order in reverse mode.
  - Implementation: ReverseAD

- Applications:
  - Uncertainty quantification
  - Chemistry: exchange-correlation (XC) energy functional
Research Overview

- **Second order reverse mode**:
  - More efficient in evaluating Hessian in both complexity and memory usage in many applications.
  - Proved to be equivalent to an variance of vertex elimination on the computational graph of the gradient.

- **High order reverse mode**: [this talk]
  - High order reverse mode: evaluating derivative tensor $\nabla^d f$ up to any order in reverse mode.
  - Implementation: ReverseAD.

- **Applications**:
  - Uncertainty quantification.
  - Chemistry: exchange-correlation (XC) energy functional.
For a scalar objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$

First order AD:
- Forward: $[F_1 \circ f](x, \dot{x}) = \sum_i \frac{\partial f}{\partial x_i} \dot{x}_i = \nabla f^T \cdot \dot{x}$
- Reverse: $[R_1 \circ f](x) = (\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_1}) = \nabla f$

Second order AD:
- (Pure) Forward: $[F_2 \circ f](x, \dot{x}) = \frac{1}{2} \dot{x}^T \cdot \nabla^2 f \cdot \dot{x}$
- Mixed: $[R_1 \circ F_1 \circ f](x, \dot{x}) = \nabla^2 f \cdot \dot{x}$
- (Pure) Reverse: $[R_2 \circ f](x) = \nabla^2 f$

High order AD:
- (Pure) Forward: High order taylor coefficients
- (Pure) Reverse: High order reverse mode
- Mixed modes then can be generated
For a scalar objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$

First order AD:
- **Forward**: $[\mathcal{F}_1 \circ f](x, \dot{x}) = \sum_i \frac{\partial f}{\partial x_i} \dot{x}_i = \nabla f^T \cdot \dot{x}$
- **Reverse**: $[\mathcal{R}_1 \circ f](x) = (\frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_1}) = \nabla f$

Second order AD:
- **(Pure) Forward**: $[\mathcal{F}_2 \circ f](x, \dot{x}) = \frac{1}{2} \dot{x}^T \cdot \nabla^2 f \cdot \dot{x}$
- **Mixed**: $[\mathcal{R}_1 \circ \mathcal{F}_1 \circ f](x, \dot{x}) = \nabla^2 f \cdot \dot{x}$
- **(Pure) Reverse**: $[\mathcal{R}_2 \circ f](x) = \nabla^2 f$

High order AD:
- **(Pure) Forward**: High order taylor coefficients
- **(Pure) Reverse**: High order reverse mode
- Mixed modes then can be generated
For a scalar objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$

**First order AD:**
- Forward: $[\mathcal{F}_1 \circ f](\mathbf{x}, \dot{\mathbf{x}}) = \sum_i \frac{\partial f}{\partial x_i} \dot{x}_i = \nabla f^T \cdot \dot{\mathbf{x}}$
- Reverse: $[\mathcal{R}_1 \circ f](\mathbf{x}) = (\frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_1}) = \nabla f$

**Second order AD:**
- (Pure) Forward: $[\mathcal{F}_2 \circ f](\mathbf{x}, \dot{\mathbf{x}}) = \frac{1}{2} \dot{\mathbf{x}}^T \cdot \nabla^2 f \cdot \dot{\mathbf{x}}$
- Mixed: $[\mathcal{R}_1 \circ \mathcal{F}_1 \circ f](\mathbf{x}, \dot{\mathbf{x}}) = \nabla^2 f \cdot \dot{\mathbf{x}}$
- (Pure) Reverse: $[\mathcal{R}_2 \circ f](\mathbf{x}) = \nabla^2 f$

**High order AD:**
- (Pure) Forward: High order taylor coefficients
- (Pure) Reverse: High order reverse mode
- Mixed modes then can be generated
For a scalar objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$

**First order AD:**
- **Forward:** $[\mathcal{F}_1 \circ f](\mathbf{x}, \dot{\mathbf{x}}) = \sum_i \frac{\partial f}{\partial x_i} \dot{x}_i = \nabla f^T \cdot \dot{\mathbf{x}}$
- **Reverse:** $[\mathcal{R}_1 \circ f](\mathbf{x}) = \left( \frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_1} \right) = \nabla f$

**Second order AD:**
- **(Pure) Forward:** $[\mathcal{F}_2 \circ f](\mathbf{x}, \dot{\mathbf{x}}) = \frac{1}{2} \dot{\mathbf{x}}^T \cdot \nabla^2 f \cdot \dot{\mathbf{x}}$
- **Mixed:** $[\mathcal{R}_1 \circ \mathcal{F}_1 \circ f](\mathbf{x}, \dot{\mathbf{x}}) = \nabla^2 f \cdot \dot{\mathbf{x}}$
- **(Pure) Reverse:** $[\mathcal{R}_2 \circ f](\mathbf{x}) = \nabla^2 f$

**High order AD:**
- **(Pure) Forward:** High order taylor coefficients
- **(Pure) Reverse:** High order reverse mode
- Mixed modes then can be generated
Background

- For a scalar objective function $f : \mathcal{R}^n \rightarrow \mathcal{R}$
  - First order AD:
    - Forward: $[\mathcal{F}_1 \circ f](x, \dot{x}) = \sum_i \frac{\partial f}{\partial x_i} \dot{x}_i = \nabla f^T \cdot \dot{x}$
    - Reverse: $[\mathcal{R}_1 \circ f](x) = (\frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_1}) = \nabla f$
  - Second order AD:
    - (Pure) Forward: $[\mathcal{F}_2 \circ f](x, \dot{x}) = \frac{1}{2} \dot{x}^T \cdot \nabla^2 f \cdot \dot{x}$
    - Mixed: $[\mathcal{R}_1 \circ \mathcal{F}_1 \circ f](x, \dot{x}) = \nabla^2 f \cdot \dot{x}$
    - (Pure) Reverse: $[\mathcal{R}_2 \circ f](x) = \nabla^2 f$
  - High order AD:
    - (Pure) Forward: High order taylor coefficients
    - (Pure) Reverse: High order reverse mode
    - Mixed modes then can be generated
For a scalar objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$

**First order AD:**
- **Forward:** $[\mathcal{F}_1 \circ f](\mathbf{x}, \dot{x}) = \sum_i \frac{\partial f}{\partial x_i} \dot{x}_i = \nabla f^T \cdot \dot{x}$
- **Reverse:** $[\mathcal{R}_1 \circ f](\mathbf{x}) = \left( \frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_1} \right) = \nabla f$

**Second order AD:**
- **(Pure) Forward:** $[\mathcal{F}_2 \circ f](\mathbf{x}, \dot{x}) = \frac{1}{2} \dot{x}^T \cdot \nabla^2 f \cdot \dot{x}$
- **Mixed:** $[\mathcal{R}_1 \circ \mathcal{F}_1 \circ f](\mathbf{x}, \dot{x}) = \nabla^2 f \cdot \dot{x}$
- **(Pure) Reverse:** $[\mathcal{R}_2 \circ f](\mathbf{x}) = \nabla^2 f$

**High order AD:**
- **(Pure) Forward:** High order taylor coefficients
- **(Pure) Reverse:** High order reverse mode
- **Mixed modes then can be generated**
For a scalar objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$

**First order AD:**
- Forward: $[F_1 \circ f](x, \dot{x}) = \sum_i \frac{\partial f}{\partial x_i} \dot{x}_i = \nabla f^T \cdot \dot{x}$
- Reverse: $[R_1 \circ f](x) = \left( \frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_1} \right) = \nabla \dot{f}$

**Second order AD:**
- (Pure) Forward: $[F_2 \circ f](x, \dot{x}) = \frac{1}{2} \dot{x}^T \cdot \nabla^2 f \cdot \dot{x}$
- Mixed: $[R_1 \circ F_1 \circ f](x, \dot{x}) = \nabla^2 f \cdot \dot{x}$
- (Pure) Reverse: $[R_2 \circ f](x) = \nabla^2 \dot{f}$

**High order AD:**
- (Pure) Forward: High order taylor coefficients
- (Pure) Reverse: High order reverse mode
- Mixed modes then can be generated
Accumulate high order taylor coefficients\(^1\):

- \(\nabla^d f\) : \(d\)-th order derivative tensor (symmetric).
- \(\nabla^d f \cdot \dot{x}\) : A tensor-vector product, \((d - 1)\)-th order symmetric tensor
- ... 
- \(\left[\left[\nabla^d f \cdot \dot{x}\right] \cdot \ddot{x}\right] \cdots \right) \cdot \dddot{x} : A scalar, the \(d\)-th order taylor coefficients.

Accumulate high order taylor coefficients

\[ \nabla^d f : d\text{-th order derivative tensor (symmetric).} \]
\[ \nabla^d f \cdot \dot{x} : A \text{ tensor-vector product, } (d - 1)\text{-th order symmetric tensor} \]
\[ \left[ \left[ \nabla^d f \cdot \dot{x} \right] \cdot \dot{x} \right] \cdots \cdot \dot{x} : A \text{ scalar, the } d\text{-th order taylor coefficients.} \]

\[ d = 2: \]
\[ [F_2 \circ f](\mathbf{x}, \dot{\mathbf{x}}) = \frac{1}{2} \mathbf{x}^T \cdot \nabla^2 f \cdot \dot{\mathbf{x}} \]
\[ [\nabla^2 f]_{ij} = [F_2 \circ f](\mathbf{x}, e_i + e_j) - [F_2 \circ f](\mathbf{x}, e_i) - [F_2 \circ f](\mathbf{x}, e_j) \]

---

Accumulate high order taylor coefficients\(^1\):

- \(\nabla^d f\): \(d\)-th order derivative tensor (symmetric).
- \(\nabla^d f \cdot \dot{x}\): A tensor-vector product, \((d - 1)\)-th order symmetric tensor
- \[
\left(\left[\nabla^d f \cdot \dot{x}\right] \cdot \dot{x}\right) \cdots \cdot \dot{x}\]: A scalar, the \(d\)-th order taylor coefficients.

General case:

- \([\mathcal{F}_d \circ f](\mathbf{x}, \dot{x}) = \frac{1}{d!} \left[\left[\nabla^d f \cdot \dot{x}\right] \cdot \dot{x}\right] \cdots \cdot \dot{x}\]
- \([\nabla^d f]_{i_1 \ldots i_d}\): a linear combination of

\[\{[\mathcal{F}_d \circ f](\mathbf{x}, \mathbf{e}) : \mathbf{e} \in \text{Span}\{e_{i_1}, \ldots, e_{i_d}\}\}\]
Accumulate high order taylor coefficients\(^1\):

- \(\nabla^d f\): \(d\)-th order derivative tensor (symmetric).
- \(\nabla^d f \cdot \dot{x}\): A tensor-vector product, \((d - 1)\)-th order symmetric tensor.
- \(\left[\left[\left[\nabla^d f \cdot \dot{x}\right] \cdot \dot{x}\right] \cdots \right] \cdot \dot{x}\): A scalar, the \(d\)-th order taylor coefficients.

General case:

- \(\mathcal{F}_d \circ f(x, \dot{x}) = \frac{1}{d!} \left[\left[\left[\nabla^d f \cdot \dot{x}\right] \cdot \dot{x}\right] \cdots \right] \cdot \dot{x}\)
- \(\nabla^d f_{i_1 \cdots i_d}\): a linear combination of \(\{\mathcal{F}_d \circ f(x, \dot{e}) : \dot{e} \in \text{Span}\{e_{i_1}, \cdots, e_{i_d}\}\}\)

Complexity: \(O\left(\binom{n+d-1}{d} \cdot l\right)\)

---

Definition

After process the SAC $v_i = \varphi_i(v_j)_{\{v_j: v_j \prec v_i\}}$ in reverse mode, the process SACs define an equivalent function $f_i(S_i)$. The objective function is the composition of $f_i$ and the remaining SACs and $S_i$ is the current live variable set.

Observation

reverse mode computes the derivatives of $f_i(S_i)$ in each step by following the order chain rule.
Reverse Mode Revisit

Definition

After process the SAC $v_i = \varphi_i(v_j)_{v_j \prec v_i}$ in reverse mode, the process SACs define an equivalent function $f_i(S_i)$. The objective function is the composition of $f_i$ and the remaining SACs and $S_i$ is the current live variable set.

Observation

Second order reverse mode computes the first and the second order derivatives of $f_i(S_i)$ in each step by following the first and second order chain rule.
**Definition**

After process the SAC $v_i = \varphi_i(v_j)\{v_j:v_j \prec v_i\}$ in reverse mode, the process SACs define an equivalent function $f_i(S_i)$. The objective function is the composition of $f_i$ and the remaining SACs and $S_i$ is the current live variable set.

**Observation**

Second order High order reverse mode computes the first and the second order derivatives up to order $d$ of $f_i(S_i)$ in each step by following the first and second high order chain rule.
High Order Chain Rule

Observation

High order reverse mode computes the derivatives up to order $d$ of $f_i$ in each step by following the high order chain rule.

- When process $v_i = \varphi_i(v_j)\{v_j: v_j \prec v_i\}$:
  - $S_i = S_{i+1} \setminus \{v_i\} \cup \{v_j : v_j \prec v_i\}$
  - $f_i(S_i) = f_{i+1}(S_{i+1} \setminus \{v_i\}, v_i = \varphi_i(v_j)\{v_j: v_j \prec v_i\})$

- High order chain rule:
  
  derivatives of $f_{i+1}(S_{i+1}) \rightarrow$ derivatives of $f_i(S_i)$

- General case of Faà di Bruno equation
- Special case of the equation in Ma, 2009

---

Observation

High order reverse mode computes the derivatives up to order $d$ of $f_i$ in each step by following the high order chain rule.

- When process $v_i = \varphi_i(v_j)\{v_j:v_j \prec v_i\}$:
  - $S_i = S_{i+1} \setminus \{v_i\} \cup \{v_j : v_j \prec v_i\}$
  - $f_i(S_i) = f_{i+1}(S_{i+1} \setminus \{v_i\})$, $v_i = \varphi_i(v_j)\{v_j:v_j \prec v_i\}$

- High order chain rule:
  
  derivatives of $f_{i+1}(S_{i+1}) \rightarrow$ derivatives of $f_i(S_i)$

- General case of Faà di Bruno equation
- Special case of the equation in Ma, 2009²

Multiset

A *multiset* $D$ is a generalization of the notion of a set in which members are allowed to appear more than once.

We use $\mathcal{D}_S$ to represent the family of all multisets over $S$. That is: $\mathcal{D}_S = \{D : D = \{e_1, e_2, \cdots, e_d\}, e_i \in S, 1 \leq i \leq d\}$

Derivative Mapping

For a function $f(S)$, its order $d$ derivative tensor can be represented as a mapping from $D \in \mathcal{D}_S, |D| = d$ to $\mathcal{R}$ as:

$$T_f(D) = \frac{\partial |D| f}{\partial D} = \frac{\partial |D| f}{\partial v_{i_1} \partial v_{i_2} \cdots \partial v_{i_{|D|}}}$$
Multiset

A *multiset* $D$ is a generalization of the notion of a set in which members are allowed to appear more than once.

We use $\mathcal{D}_S$ to represent the family of all multisets over $S$. That is: $\mathcal{D}_S = \{D : D = \{e_1, e_2, \cdots, e_d\}, e_i \in S, 1 \leq i \leq d\}$

Derivative Mapping

For a function $f(S)$, its order $d$ derivative tensor can be represented as a mapping from $D \in \mathcal{D}_S, |D| = d$ to $\mathcal{R}$ as:

$$T_f(D) = \frac{\partial^{|D|}f}{\partial D} = \frac{\partial^{|D|}f}{\partial v_{i_1} \partial v_{i_2} \cdots \partial v_{i_{|D|}}}$$
Multiset

A *multiset* $D$ is a generalization of the notion of a set in which members are allowed to appear more than once.

We use $\mathcal{D}_S$ to represent the family of all multisets over $S$. That is: $\mathcal{D}_S = \{D : D = \{e_1, e_2, \cdots, e_d\}, e_i \in S, 1 \leq i \leq d\}$

Derivative Mapping

For a function $f(S)$, its order $d$ derivative tensor can be represented as a mapping from $D \in \mathcal{D}_S, |D| = d$ to $\mathbb{R}$ as:

$$T_f(D) = \frac{\partial |D| f}{\partial D} = \frac{\partial |D| f}{\partial v_{i_1} \partial v_{i_2} \cdots \partial v_{i_{|D|}}}$$
\[ T_{f_i}(D) = T_{f_{i+1}}(D) + \sum_{D_L \notin D} \left[ \sum_{D_i \cap D_j = \emptyset} \frac{\partial \varphi_i}{\partial D_1} \cdots \frac{\partial \varphi_i}{\partial D_r} T_{f_{i+1}}\left(D_L \cup \{v_i^f\}\right) \right] \]

- \( D_L, D_1, \ldots, D_r \) is a partition of \( D \).
- First order: \( D = \{v\} \)
  - \( a_i(v) = a_{i+1}(v) + \frac{\partial \varphi_i}{\partial v} a_{i+1}(v_i) \)
  - \( \frac{\partial \varphi_i}{\partial v} a_{i+1}(v_i) : D_L = \emptyset, D_1 = \{v\} \)
\[ T_{f_i}(D) = T_{f_{i+1}}(D) + \sum_{D_L \not\subset D} \left[ \sum_{D_i \cap D_j = \emptyset} \frac{\partial \varphi_i}{\partial D_1} \cdots \frac{\partial \varphi_i}{\partial D_r} T_{f_{i+1}}(D_L \cup \{v_i^f\}) \right] \]
High Order Chain Rule

\[ T_{f_i}(D) = T_{f_{i+1}}(D) + \sum_{D_L \not\subset D} \left[ \sum_{D_i \cap D_j = \emptyset} \frac{\partial \varphi_i}{\partial D_1} \cdots \frac{\partial \varphi_i}{\partial D_r} T_{f_{i+1}}(D_L \cup \{v_i\}) \right] \]

- \( D_L, D_1, \ldots, D_r \) is a partition of \( D \).
- First order: \( D = \{v\} \)
  - \( a_i(v) = a_{i+1}(v) + \frac{\partial \varphi_i}{\partial v} a_{i+1}(v_i) \)
  - \( \frac{\partial \varphi_i}{\partial v} a_{i+1}(v_i) : D_L = \emptyset, D_1 = \{v\} \)
\[ T_{f_i}(D) = T_{f_{i+1}}(D) + \sum_{D_L \not\subseteq D} \left[ \sum_{D_i \cap D_j = \emptyset \atop D_1 \cup \cdots \cup D_r = D \setminus D_L} \frac{\partial \varphi_i}{\partial D_1} \cdots \frac{\partial \varphi_i}{\partial D_r} T_{f_{i+1}}(D_L \cup \{v_i\}) \right] \]

- \( D_L, D_1, \cdots, D_r \) is a partition of \( D \).
- First order: \( D = \{v\} \)
  
  - \( a_i(v) = a_{i+1}(v) + \frac{\partial \varphi_i}{\partial v} a_{i+1}(v_i) \)
  
  - \( \frac{\partial \varphi_i}{\partial v} a_{i+1}(v_i) : D_L = \emptyset, D_1 = \{v\} \)
\[ \mathcal{T}_{f_i}(D) = \mathcal{T}_{f_{i+1}}(D) + \sum_{D_L \not\subset D} \left[ \sum_{D_i \cap D_j = \emptyset} \sum_{D_1 \cup \cdots \cup D_r = D \setminus D_L} \frac{\partial \varphi_i}{\partial D_1} \cdots \frac{\partial \varphi_i}{\partial D_r} \mathcal{T}_{f_{i+1}}(D_L \cup \{v_i\}) \right] \]

- \( D_L, D_1, \cdots, D_r \) is a partition of \( D \).
- First order: \( D = \{v\} \)
  - \( a_i(v) = a_{i+1}(v) + \frac{\partial \varphi_i}{\partial v} a_{i+1}(v_i) \)
  - \( \frac{\partial \varphi_i}{\partial v} a_{i+1}(v_i) : D_L = \emptyset, D_1 = \{v\} \)
\[ T_{f_i}(D) = T_{f_{i+1}}(D) + \sum_{D_L \subseteq D} \left[ \sum_{\begin{array}{c} D_i \cap D_j = \emptyset \\ D_1 \cup \cdots \cup D_r = D \setminus D_L \end{array}} \frac{\partial \varphi_i}{\partial D_1} \cdots \frac{\partial \varphi_i}{\partial D_r} T_{f_{i+1}}(D_L \cup \{v_i^f\}) \right] \]

\[ \begin{align*}
\text{DL, D1, \cdots, Dr is a partition of D.} \\
\text{Second order: } D = \{v, u\} & \\

h_i(v, u) &= h_{i+1}(v, u) + \frac{\partial \varphi_i}{\partial v} h_{i+1}(v_i, u) + \frac{\partial \varphi_i}{\partial u} h_{i+1}(v, v_i) \\
&\quad + \frac{\partial \varphi_i}{\partial v} \frac{\partial \varphi_i}{\partial u} h_{i+1}(v_i, v_i) + \frac{\partial^2 \varphi_i}{\partial v \partial u} a_{i+1}(v_i)
\end{align*} \]
\( T_{f_i}(D) = T_{f_{i+1}}(D) + \sum_{D_L \notin D} \left[ \sum_{D_i \cap D_j = \emptyset \atop D_1 \cup \cdots \cup D_r = D \setminus D_L} \frac{\partial \varphi_i}{\partial D_1} \cdots \frac{\partial \varphi_i}{\partial D_r} T_{f_{i+1}}(D_L \cup \{v_i^f\}) \right] \)

- \( D_L, D_1, \cdots, D_r \) is a partition of \( D \).
- Second order: \( D = \{v, u\} \)

\[
h_i(v, u) = h_{i+1}(v, u) + \frac{\partial \varphi_i}{\partial v} h_{i+1}(v_i, u) + \frac{\partial \varphi_i}{\partial u} h_{i+1}(v, v_i) \\
+ \frac{\partial \varphi_i}{\partial v} \frac{\partial \varphi_i}{\partial u} h_{i+1}(v_i, v_i) + \frac{\partial^2 \varphi_i}{\partial v \partial u} a_{i+1}(v_i)
\]
High Order Chain Rule

\[ T_{f_i}(D) = T_{f_{i+1}}(D) + \sum_{D_L \not\subseteq D} \left[ \sum_{D_i \cap D_j = \emptyset} \frac{\partial \varphi_i}{\partial D_1} \cdots \frac{\partial \varphi_i}{\partial D_r} T_{f_{i+1}}(D_L \cup \{v_i^f\}) \right] \]

- \( D_L, D_1, \cdots, D_r \) is a partition of \( D \).
- Second order: \( D = \{v, u\} \)

\[ h_i(v, u) = h_{i+1}(v, u) + \frac{\partial \varphi_i}{\partial v} h_{i+1}(v_i, u) + \frac{\partial \varphi_i}{\partial u} h_{i+1}(v, v_i) \]
\[ + \frac{\partial \varphi_i}{\partial v} \frac{\partial \varphi_i}{\partial u} h_{i+1}(v_i, v_i) + \frac{\partial^2 \varphi_i}{\partial v \partial u} a_{i+1}(v_i) \]

- \( \frac{\partial \varphi_i}{\partial v} h_{i+1}(v_i, u) : D_L = \{u\}, D_1 = \{v\} \)
High Order Chain Rule

$$T_f_i(D) = T_{f_{i+1}}(D) + \sum_{D_L \neq D} \left[ \sum_{D_i \cap D_j = \emptyset} \frac{\partial \varphi_i}{\partial D_1} \cdots \frac{\partial \varphi_i}{\partial D_r} T_{f_{i+1}}(D_L \cup \{v_i^f\}) \right]$$

- $D_L, D_1, \ldots, D_r$ is a partition of $D$.
- Second order: $D = \{v, u\}$

$$h_i(v, u) = h_{i+1}(v, u) + \frac{\partial \varphi_i}{\partial v} h_{i+1}(v_i, u) + \frac{\partial \varphi_i}{\partial u} h_{i+1}(v, v_i)$$

$$+ \frac{\partial \varphi_i}{\partial v} \frac{\partial \varphi_i}{\partial u} h_{i+1}(v_i, v_i) + \frac{\partial^2 \varphi_i}{\partial v \partial u} a_{i+1}(v_i)$$

- $\frac{\partial \varphi_i}{\partial u} h_{i+1}(v, v_i) : D_L = \{v\}, D_1 = \{u\}$
High Order Chain Rule

\[ T_{f_i}(D) = T_{f_{i+1}}(D) + \sum_{D_L \not\in \mathcal{D}} \left[ \sum_{D_i \cap D_j = \emptyset} \left( \frac{\partial \varphi_i}{\partial D_1} \cdots \frac{\partial \varphi_i}{\partial D_r} T_{f_{i+1}}(D_L \cup \{v_i\}) \right) \right] \]

- \( D_L, D_1, \ldots, D_r \) is a partition of \( D \).
- Second order: \( D = \{v, u\} \)

\[
h_i(v, u) = h_{i+1}(v, u) + \frac{\partial \varphi_i}{\partial v} h_{i+1}(v_i, u) + \frac{\partial \varphi_i}{\partial u} h_{i+1}(v, v_i) + \frac{\partial \varphi_i}{\partial v} \frac{\partial \varphi_i}{\partial u} h_{i+1}(v_i, v_i) + \frac{\partial^2 \varphi_i}{\partial v \partial u} a_{i+1}(v_i)
\]

- \( \frac{\partial \varphi_i}{\partial v} \frac{\partial \varphi_i}{\partial u} h_{i+1}(v_i, v_i) : D_L = \emptyset, D_1 = \{v\}, D_2 = \{u\} \)
\[ T_{f_i}(D) = T_{f_{i+1}}(D) + \sum_{D_L \not\subseteq D} \left[ \sum_{D_i \cap D_j = \emptyset} \frac{\partial \varphi_i}{\partial D_1} \cdots \frac{\partial \varphi_i}{\partial D_r} T_{f_{i+1}}(D_L \cup \{v_i\}) \right] \]

- \( D_L, D_1, \cdots, D_r \) is a partition of \( D \).
- Second order: \( D = \{v, u\} \)

\[ h_i(v, u) = h_{i+1}(v, u) + \frac{\partial \varphi_i}{\partial v} h_{i+1}(v_i, u) + \frac{\partial \varphi_i}{\partial u} h_{i+1}(v, v_i) \]
\[ + \frac{\partial \varphi_i}{\partial v} \frac{\partial \varphi_i}{\partial u} h_{i+1}(v_i, v_i) + \frac{\partial^2 \varphi_i}{\partial v \partial u} a_{i+1}(v_i) \]

- \( \frac{\partial^2 \varphi_i}{\partial v \partial u} a_{i+1}(v_i) : D_L = \emptyset, \ D_1 = \{v, u\} \)
\[ T_{f_i}(D) = T_{f_{i+1}}(D) + \sum_{D_L \notin D} \left[ \sum_{D_i \cap D_j = \emptyset \atop D_1 \cup \cdots \cup D_r = D \setminus D_L} \frac{\partial \varphi_i}{\partial D_1} \cdots \frac{\partial \varphi_i}{\partial D_r} T_{f_{i+1}}(D_L \cup \{v_i^f\}) \right] \]

- \( D_L, D_1, \ldots, D_r \) is a partition of \( D \).
- \( B_{d+1} \) summations: \((d + 1)\) th Bell number.
- \( O(B_{d+1} \cdot s_i^{d-1}) \) updates for each SAC.
- Overall complexity: \( O(B_{d+1} \cdot s^{d-1} \cdot l) \), \( s = \max\{s_i\} \)
  - When \( d = 1 \): \( O(l) \) Baur-Strassen theorem.
  - When \( d = 2 \): \( O(l \cdot s) \) second order reverse mode
  - When \( d = 3 \): \( O(l \cdot s^2) \) third order reverse mode
  - ...
\[ T_f(D) = T_{f+1}(D) + \sum_{D_L \notin D} \left[ \sum_{D_i \cap D_j = \emptyset} \frac{\partial \varphi_i}{\partial D_1} \cdots \frac{\partial \varphi_i}{\partial D_r} T_{f+1}(D_L \cup \{v_i\}) \right] \]

- \( D_L, D_1, \ldots, D_r \) is a partition of \( D \).
- \( B_{d+1} \) summations: \( (d+1) \) th Bell number.
- \( O(B_{d+1} \cdot s^{d-1}) \) updates for each SAC.
- Overall complexity: \( O(B_{d+1} \cdot s^{d-1} \cdot l) \), \( s = \max\{s_i\} \)
  - When \( d = 1 \): \( O(l) \) Baur-Strassen theorem.
  - When \( d = 2 \): \( O(l \cdot s) \) second order reverse mode
  - When \( d = 3 \): \( O(l \cdot s^2) \) third order reverse mode
  - \( \ldots \)
High Order Reverse Mode: Complexity

\[ T_{f_i}(D) = T_{f_{i+1}}(D) + \sum_{D_L \notin D} \left[ \sum_{D_i \cap D_j = \emptyset \atop D_1 \cup \ldots \cup D_r = D \setminus D_L} \frac{\partial \varphi_i}{\partial D_1} \cdots \frac{\partial \varphi_i}{\partial D_r} T_{f_{i+1}}(D_L \cup \{v_i^f\}) \right] \]

- \( D_L, D_1, \ldots, D_r \) is a partition of \( D \).
- \( B_{d+1} \) summations: \((d + 1)\) th Bell number.
- \( O(B_{d+1} \cdot s_i^{d-1}) \) updates for each SAC.
- Overall complexity: \( O(B_{d+1} \cdot s^{d-1} \cdot l) \), \( s = \max\{s_i\} \)
  - When \( d = 1 \): \( O(l) \) Baur-Strassen theorem.
  - When \( d = 2 \): \( O(l \cdot s) \) second order reverse mode
  - When \( d = 3 \): \( O(l \cdot s^2) \) third order reverse mode
  - \( \ldots \)
High Order Reverse Mode: Complexity

\[ T_{f_i}(D) = T_{f_{i+1}}(D) + \sum_{D_L \not\subseteq D} \left[ \sum_{D_i \cap D_j = \emptyset} \sum_{D_1 \cup \ldots \cup D_r = D \setminus D_L} \frac{\partial \varphi_i}{\partial D_1} \cdots \frac{\partial \varphi_i}{\partial D_r} T_{f_{i+1}}(D_L \cup \{v_i^f \}) \right] \]

- \( D_L, D_1, \ldots, D_r \) is a partition of \( D \).
- \( B_{d+1} \) summations: \((d + 1)\) th Bell number.
- \( O(B_{d+1} \cdot s_i^{d-1}) \) updates for each SAC.
- Overall complexity: \( O(B_{d+1} \cdot s^{d-1} \cdot l) \), \( s = \max\{s_i\} \)
  - When \( d = 1 \): \( O(l) \) Baur-Strassen theorem.
  - When \( d = 2 \): \( O(l \cdot s) \) second order reverse mode
  - When \( d = 3 \): \( O(l \cdot s^2) \) third order reverse mode
  - \( \ldots \)
\[ T_{f_i}(D) = T_{f_{i+1}}(D) + \sum_{D_L \not\in D} \left[ \sum_{D_i \cap D_j = \emptyset} \frac{\partial \varphi_i}{\partial D_1} \cdots \frac{\partial \varphi_i}{\partial D_r} T_{f_{i+1}}(D_L \cup \{v_i^f\}) \right] \]

- \( D_L, D_1, \cdots, D_r \) is a partition of \( D \).
- \( B_{d+1} \) summations: \((d + 1)\) th Bell number.
- \( O(B_{d+1} \cdot s_i^{d-1}) \) updates for each SAC.
- Overall complexity: \( O(B_{d+1} \cdot s^{d-1} \cdot l) \), \( s = \max\{s_i\} \)
  - When \( d = 1 \): \( O(l) \) Baur-Strassen theorem.
  - When \( d = 2 \): \( O(l \cdot s) \) second order reverse mode
  - When \( d = 3 \): \( O(l \cdot s^2) \) third order reverse mode
\[ T_{f_i}(D) = T_{f_{i+1}}(D) + \sum_{D_L \notin D} \sum_{D_i \cap D_j = \emptyset} \frac{\partial \phi_i}{\partial D_1} \cdots \frac{\partial \phi_i}{\partial D_r} T_{f_{i+1}}(D_L \cup \{v_i^f\}) \]

- \( D_L, D_1, \ldots, D_r \) is a partition of \( D \).
- \( B_{d+1} \) summations : \((d + 1)\) th Bell number.
- \( O(B_{d+1} \cdot s_i^{d-1}) \) updates for each SAC.
- Overall complexity : \( O(B_{d+1} \cdot s^{d-1} \cdot l), \ s = \max\{s_i\} \)
  - When \( d = 1 \) : \( O(l) \) Baur-Strassen theorem.
  - When \( d = 2 \) : \( O(l \cdot s) \) second order reverse mode
  - When \( d = 3 \) : \( O(l \cdot s^2) \) third order reverse mode
  - ...
\[ T_f(D) = T_{f+1}(D) + \sum_{D_L \notin D} \left[ \sum_{D_i \cap D_j = \emptyset} \sum_{D_1 \cup \cdots \cup D_r = D \setminus D_L} \frac{\partial \varphi_i}{\partial D_1} \cdots \frac{\partial \varphi_i}{\partial D_r} T_{f+1}(D_L \cup \{v_i^f\}) \right] \]

- \( D_L, D_1, \cdots, D_r \) is a partition of \( D \).
- \( B_{d+1} \) summations: \((d + 1)\) th Bell number.
- \( O(B_{d+1} \cdot s_i^{d-1}) \) updates for each SAC.
- Overall complexity: \( O(B_{d+1} \cdot s^{d-1} \cdot l) \), \( s = \max\{s_i\} \)
  - When \( d = 1 \): \( O(l) \) Baur-Strassen theorem.
  - When \( d = 2 \): \( O(l \cdot s) \) second order reverse mode
  - When \( d = 3 \): \( O(l \cdot s^2) \) third order reverse mode
  - ...
$T_{f_i}(D) = T_{f_{i+1}}(D) + \sum_{D_L \not\subset D} \left[ \sum_{D_i \cap D_j = \emptyset} \frac{\partial \varphi_i}{\partial D_1} \cdots \frac{\partial \varphi_i}{\partial D_r} T_{f_{i+1}}(D_L \cup \{v^i_r\}) \right]$}

- $D_L, D_1, \cdots, D_r$ is a partition of $D$.
- $B_{d+1}$ summations: $(d + 1)$ th Bell number.
- $O(\ B_{d+1} \cdot s_i^{d-1})$ updates for each SAC.
- Overall complexity: $O(\ B_{d+1} \cdot s^{d-1} \cdot l), \ s = \max \{s_i\}$
  - When $d = 1$: $O(l)$ Baur-Strassen theorem.
  - When $d = 2$: $O(l \cdot s)$ second order reverse mode
  - When $d = 3$: $O(l \cdot s^2)$ third order reverse mode
  - ...
\[ T_{f_i}(D) = T_{f_{i+1}}(D) + \sum_{D_L \notin D} \left[ \sum_{D_i \cap D_j = \emptyset, D_1 \cup \ldots \cup D_r = D \setminus D_L} \frac{\partial \varphi_i}{\partial D_1} \cdots \frac{\partial \varphi_i}{\partial D_r} T_{f_{i+1}}(D_L \cup \{v^i_r\}) \right] \]

- \( D_L, D_1, \ldots, D_r \) is a partition of \( D \)
- Generate all \( D_L \), s.t, \( T_{f_{i+1}}(D_L \cup \{v^i_r\}) \neq 0 \) and \( D_1, \ldots, D_r \), s.t, \( \frac{\partial \varphi_i}{\partial D_i} \neq 0, 1 \leq i \leq r \)
- Then perform incremental updates on \( D = D_L \cup D_1 \cup \cdots \cup D_r \)
- More than one way to partition \( D \) into \( D_L, D_1, \ldots, D_r \).
  - SymCoeff(\( D_L, D_1, \ldots, D_r \)) : Multiplicity that partition \( D \) into \( D_L, D_1, \ldots, D_r \).
  - Flat code for pre-computed symmetric coefficients
  - \( \sim 5k \) lines of generated code for up to sixth order
\[ T_{f_i}(D) = T_{f_{i+1}}(D) + \sum_{D_L \notin D} \left[ \sum_{D_i \cap D_j = \emptyset} \frac{\partial \varphi_i}{\partial D_1} \cdots \frac{\partial \varphi_i}{\partial D_r} T_{f_{i+1}}(D_L \cup \{v_i^f\}) \right] \]

- \( D_L, D_1, \cdots, D_r \) is a partition of \( D \)
- Generate all \( D_L \), s.t, \( T_{f_{i+1}}(D_L \cup \{v_i^f\}) \neq 0 \) and \( D_1, \cdots, D_r \), s.t, \( \frac{\partial \varphi_i}{\partial D_i} \neq 0, 1 \leq i \leq r \)
- Then perform incremental updates on \( D = D_L \cup D_1 \cup \cdots \cup D_r \)
- More than one way to partition \( D \) into \( D_L, D_1, \cdots, D_r \).
  - \( \text{SymCoeff}(D_L, D_1, \cdots, D_r) \) : Multiplicity that partition \( D \) into \( D_L, D_1, \cdots, D_r \).
  - Flat code for pre-computed symmetric coefficients
  - \( \sim 5k \) lines of generated code for up to sixth order
\[ T_{f_i}(D) = T_{f_{i+1}}(D) + \sum_{D_L \notin D} \left[ \sum_{D_i \cap D_j = \emptyset} \frac{\partial \varphi_i}{\partial D_1} \cdots \frac{\partial \varphi_i}{\partial D_r} T_{f_{i+1}}(D_L \cup \{v_r^i\}) \right] \]

- \( D_L, D_1, \cdots, D_r \) is a partition of \( D \)
- Generate all \( D_L \), s.t, \( T_{f_{i+1}}(D_L \cup \{v_r^i\}) \neq 0 \) and \( D_1, \cdots, D_r \), s.t, \( \frac{\partial \varphi_i}{\partial D_i} \neq 0, 1 \leq i \leq r \)
- Then perform incremental updates on \( D = D_L \cup D_1 \cup \cdots \cup D_r \)
- More than one way to partition \( D \) into \( D_L, D_1, \cdots, D_r \).
  - \( \text{SymCoeff}(D_L, D_1, \cdots, D_r) \) : Multiplicity that partition \( D \) into \( D_L, D_1, \cdots, D_r \).
  - Flat code for pre-computed symmetric coefficients
  - \( \sim 5k \) lines of generated code for up to sixth order
\[ T_{f_i}(D) = T_{f_{i+1}}(D) + \sum_{D_L \not\subset D} \left[ \sum_{D_i \cap D_j = \emptyset} \frac{\partial \varphi_i}{\partial D_1} \cdots \frac{\partial \varphi_i}{\partial D_r} T_{f_{i+1}}(D_L \cup \{ v_i^f \}) \right] \]

- \( D_L, D_1, \cdots, D_r \) is a partition of \( D \)

- Generate all \( D_L, \) s.t., \( T_{f_{i+1}}(D_L \cup \{ v_i^f \}) \neq 0 \) and \( D_1, \cdots, D_r, \) s.t., \( \frac{\partial \varphi_i}{\partial D_i} \neq 0, 1 \leq i \leq r \)

- Then perform incremental updates on \( D = D_L \cup D_1 \cup \cdots \cup D_r \)

- More than one way to partition \( D \) into \( D_L, D_1, \cdots, D_r. \)
  - \( \text{SymCoeff}(D_L, D_1, \cdots, D_r) \): Multiplicity that partition \( D \) into \( D_L, D_1, \cdots, D_r. \)
  - Flat code for pre-computed symmetric coefficients
  - \( \sim 5k \) lines of generated code for up to sixth order
\[
\mathcal{T}_{f_i}(D) = \mathcal{T}_{f_i+1}(D) + \sum_{D_L \notin D} \left[ \sum_{D_i \cap D_j = \emptyset} \frac{\partial \varphi_i}{\partial D_1} \cdots \frac{\partial \varphi_i}{\partial D_r} \mathcal{T}_{f_i+1}(D_L \cup \{v^i_r\}) \right]
\]

- \(D_L, D_1, \cdots, D_r\) is a partition of \(D\)
- Generate all \(D_L\), s.t, \(\mathcal{T}_{f_i+1}(D_L \cup \{v^i_r\}) \neq 0\) and \(D_1, \cdots, D_r\), s.t, \(\frac{\partial \varphi_i}{\partial D_i} \neq 0, 1 \leq i \leq r\)
- Then perform incremental updates on \(D = D_L \cup D_1 \cup \cdots \cup D_r\)
- More than one way to partition \(D\) into \(D_L, D_1, \cdots, D_r\).
  - \(\text{SymCoeff}(D_L, D_1, \cdots, D_r)\) : Multiplicity that partition \(D\) into \(D_L, D_1, \cdots, D_r\).
  - Flat code for pre-computed symmetric coefficients
  - \(~5k\) lines of generated code for up to sixth order
\[ T_{f_i}(D) = T_{f_{i+1}}(D) + \sum_{D_L \not\subseteq D} \left[ \sum_{D_i \cap D_j = \emptyset, D_1 \cup \cdots \cup D_r = D \setminus D_L} \frac{\partial \varphi_i}{\partial D_1} \cdots \frac{\partial \varphi_i}{\partial D_r} T_{f_{i+1}}(D_L \cup \{ v_i^f \}) \right] \]

- \( D_L, D_1, \cdots, D_r \) is a partition of \( D \)
- Generate all \( D_L \), s.t, \( T_{f_{i+1}}(D_L \cup \{ v_i^f \}) \neq 0 \) and \( D_1, \cdots, D_r \), s.t, \( \frac{\partial \varphi_i}{\partial D_i} \neq 0, 1 \leq i \leq r \)
- Then perform incremental updates on \( D = D_L \cup D_1 \cup \cdots \cup D_r \)
- More than one way to partition \( D \) into \( D_L, D_1, \cdots, D_r \).
  - \textit{SymCoeff}(D_L, D_1, \cdots, D_r) : \) Multiplicity that partition \( D \) into \( D_L, D_1, \cdots, D_r \).
  - Flat code for pre-computed symmetric coefficients
  - \( \sim 5k \) lines of generated code for up to sixth order
$T_{fi}(D) = T_{fi+1}(D) + \sum_{D_L \notin D} \left[ \sum_{D_i \cap D_j = \emptyset \atop D_1 \cup \cdots \cup D_r = D \setminus D_L} \frac{\partial \varphi_i}{\partial D_1} \cdots \frac{\partial \varphi_i}{\partial D_r} T_{fi+1}(D_L \cup \{v_i^f\}) \right]$

- $D_L, D_1, \cdots, D_r$ is a partition of $D$
- Generate all $D_L$, s.t, $T_{fi+1}(D_L \cup \{v_i^f\}) \neq 0$ and $D_1, \cdots, D_r$, s.t, $\frac{\partial \varphi_i}{\partial D_i} \neq 0, 1 \leq i \leq r$
- Then perform incremental updates on $D = D_L \cup D_1 \cup \cdots \cup D_r$
- More than one way to partition $D$ into $D_L, D_1, \cdots, D_r$
  - $\text{SymCoeff}(D_L, D_1, \cdots, D_r)$ : Multiplicity that partition $D$ into $D_L, D_1, \cdots, D_r$.
  - Flat code for pre-computed symmetric coefficients
  - $\sim 5k$ lines of generated code for up to sixth order
\[ T_{f_i}(D) = T_{f_{i+1}}(D) + \sum_{D_L \not\subseteq D} \left[ \sum_{D_i \cap D_j = \emptyset} \frac{\partial \varphi_i}{\partial D_1} \cdots \frac{\partial \varphi_i}{\partial D_r} T_{f_{i+1}}(D_L \cup \{v_i^f\}) \right] \]

- **ReverseAD**: an operator overloading implementation of the *high order reverse mode* in C++11.
  - Available at https://github.com/wangmu0701/ReverseAD.
\[ T_{f_i}(D) = T_{f_{i+1}}(D) + \sum_{D_L \notin D} \left[ \sum_{D_i \cap D_j = \emptyset} \frac{\partial \varphi_i}{\partial D_1} \cdots \frac{\partial \varphi_i}{\partial D_r} T_{f_{i+1}}(D_L \cup \{v_i^f\}) \right] \]

- **ReverseAD**: an operator overloading implementation of the *high order reverse mode* in C++11.
  - Available at [https://github.com/wangmu0701/ReverseAD](https://github.com/wangmu0701/ReverseAD).
- **Monotonic indexing** for variables on the trace
  \[ v_j \prec v_i \implies \text{index}(v_j) < \text{index}(v_i) \]
Performance: Synthetic Function

A synthetic function designed with parameters:

- $n$: number of independent variables
- $s$: size of live variables during the function evaluation
- $l$: the complexity of the function
- Dense derivatives

$$y = \sqrt{s \prod_{i=1}^{s} t_i},$$
$$t_i = ID_k \circ \cdots \circ ID_1(z_i),$$
$$z_i = t,$$
$$t = \sum_{i=1}^{n} x_i.$$

$$ID(z) = \begin{cases} 
\sqrt{z \ast z}, \\
2.0 + z - 2.0, \\
z \ast 2.0 \ast 0.5, \\
\log(\exp(z)), \\
1.0/(1.0/z), \\
sin(asin(z)). 
\end{cases}$$
A synthetic function designed with parameters:

- $n$: number of independent variables
- $s$: size of live variables during the function evaluation
- $l$: the complexity of the function
- Dense derivatives

\[
y = \sqrt{s} \prod_{i=1}^{s} t_i, \\
t_i = ID_k \circ \cdots \circ ID_1(z_i), \\
z_i = t, \\
t = \sum_{i=1}^{n} x_i.
\]

\[
ID(z) = \begin{cases} 
\sqrt{z \ast z}, \\
2.0 + z - 2.0, \\
z \ast 2.0 \ast 0.5, \\
\log(\exp(z)), \\
1.0/(1.0/z), \\
sin(asin(z)).
\end{cases}
\]
Fixed \( l \), let \( n \) and \( s \) change simultaneously.
Fixed $l$, let $n$ and $s$ change simultaneously.
Performance: Synthetic Function

- Fixed $l$ and $n$, changed $s$
Fixed \( l \) and \( n \), changed \( s \)
Performance : Synthetic Function

- Fixed $l$ and $s$, changed $n$
Fixed \( l \) and \( s \), changed \( n \)

![Bar chart showing performance comparison between ADOLC: Taylor, ReverseAD: General, ReverseAD: Special 3rd, and ReverseAD: Flat for different values of \( n \).]
Application : XCFUN (on going)

- Arbitrary order Exchange-Correlation functional library
  - https://github.com/dftlibs/xcfun
  - Using libtaylor to evaluate derivatives of functionals
  - Up to third order in current implementation
  - Small number of independents : 20 at most
  - Not so-complex functionals

- On a collection of functionals:
  - Third order Libtaylor : 81ms
  - Third order ReverseAD : 20ms
  - Fourth order ReverseAD : 83ms
Application : XCFUN (on going)

- Arbitrary order Exchange-Correlation functional library
  - https://github.com/dftlibs/xcfun
  - Using libtaylor to evaluate derivatives of functionals
  - Up to third order in current implementation
  - Small number of independents : 20 at most
  - Not so-complex functionals

- On a collection of functionals:
  - Third order Libtaylor : 81ms
  - Third order ReverseAD : 20ms
  - Fourth order ReverseAD : 83ms
High order derivative tensors could (and probably should) be directly evaluated via reverse mode.

A series of algorithms to evaluate derivatives $\mathcal{T}_f$ up to order $d$:

$\mathcal{F}_d \rightarrow \cdots \rightarrow \mathcal{F}_1 \circ \mathcal{R}_{d-1} \rightarrow \mathcal{R}_d$

- $\mathcal{R}_d$: symmetric derivative tensor $\nabla^d f$
- $\mathcal{F}_1 \circ \mathcal{R}_{d-1}$: tensor-vector $\nabla^d f \cdot \dot{x}$
- $\mathcal{F}_d$: $\left[ \left[ [\nabla^d f \cdot \dot{x}] \cdot \dot{x} \right] \cdots \right] \cdot \dot{x}$
- The structural (and sparsity) properties of $\mathcal{T}_f$ determines the optimal method.
- General compression and recovery using $\mathcal{F}_1 \circ \mathcal{R}_{d-1}$.

perfectly parallelizable
Conclusion and Future Work

- High order derivative tensors could (and probably should) be directly evaluated via reverse mode.
- A series of algorithms to evaluate derivatives $\mathcal{T}_f$ up to order $d$:

$$\mathcal{F}_d \rightarrow \cdots \rightarrow \mathcal{F}_1 \circ \mathcal{R}_{d-1} \rightarrow \mathcal{R}_d$$

- $\mathcal{R}_d$: symmetric derivative tensor $\nabla^d f$
- $\mathcal{F}_1 \circ \mathcal{R}_{d-1}$: tensor-vector $\nabla^d f \cdot \dot{x}$
- $\mathcal{F}_d$: 
  $$\left[ \left[ \nabla^d f \cdot \dot{x} \right] \cdots \right] \cdot \dot{x}$$
- The structural (and sparsity) properties of $\mathcal{T}_f$ determines the optimal method.
- General compression and recovery using $\mathcal{F}_1 \circ \mathcal{R}_{d-1}$.

  perfectly parallelizable
Conclusion and Future Work

- High order derivative tensors could (and probably should) be directly evaluated via reverse mode.

- A series of algorithms to evaluate derivatives $\mathcal{T}_f$ up to order $d$:

\[ \mathcal{F}_d \rightarrow \cdots \rightarrow \mathcal{F}_1 \circ \mathcal{R}_{d-1} \rightarrow \mathcal{R}_d \]

- $\mathcal{R}_d$: symmetric derivative tensor $\nabla^d f$
- $\mathcal{F}_1 \circ \mathcal{R}_{d-1}$: tensor-vector $\nabla^d f \cdot \dot{x}$
- $\mathcal{F}_d$: 
  
  \[
  \left[ \left[ [\nabla^d f \cdot \dot{x}] \cdot \dot{x} \right] \cdots \right] \cdot \dot{x}
  
  - The structural (and sparsity) properties of $\mathcal{T}_f$ determines the optimal method.
- General compression and recovery using $\mathcal{F}_1 \circ \mathcal{R}_{d-1}$.

  perfectly parallelizable
High order derivative tensors could (and probably should) be directly evaluated via reverse mode.

A series of algorithms to evaluate derivatives $\mathcal{T}_f$ up to order $d$:

$$\mathcal{F}_d \rightarrow \cdots \rightarrow \mathcal{F}_1 \circ \mathcal{R}_{d-1} \rightarrow \mathcal{R}_d$$

- $\mathcal{R}_d$: symmetric derivative tensor $\nabla^d f$
- $\mathcal{F}_1 \circ \mathcal{R}_{d-1}$: tensor-vector $\nabla^d f \cdot \dot{x}$
- $\mathcal{F}_d$: $\left[ \left[ \nabla^d f \cdot \dot{x} \right] \cdot \dot{x} \right] \cdots \cdot \dot{x}$

The structural (and sparsity) properties of $\mathcal{T}_f$ determines the optimal method.

General compression and recovery using $\mathcal{F}_1 \circ \mathcal{R}_{d-1}$.

perfectly parallelizable
Conclusion and Future Work

- High order derivative tensors could (and probably should) be directly evaluated via reverse mode.

- A series of algorithms to evaluate derivatives $\mathcal{T}_f$ up to order $d$:

$$\mathcal{F}_d \rightarrow \cdots \rightarrow \mathcal{F}_1 \circ \mathcal{R}_{d-1} \rightarrow \mathcal{R}_d$$

- $\mathcal{R}_d$: symmetric derivative tensor $\nabla^d f$

- $\mathcal{F}_1 \circ \mathcal{R}_{d-1}$: tensor-vector $\nabla^d f \cdot \dot{x}$

- $\mathcal{F}_d$: $\left[\left[\left[\nabla^d f \cdot \dot{x}\right] \cdot \dot{x}\right] \cdots \right] \cdot \dot{x}$

- The structural (and sparsity) properties of $\mathcal{T}_f$ determines the optimal method.

- General compression and recovery using $\mathcal{F}_1 \circ \mathcal{R}_{d-1}$.

  perfectly parallelizable
Conclusion and Future Work

- High order derivative tensors could (and probably should) be directly evaluated via reverse mode.
- A series of algorithms to evaluate derivatives $T_f$ up to order $d$:

$$F_d \rightarrow \cdots \rightarrow F_1 \circ R_{d-1} \rightarrow R_d$$

- $R_d$: symmetric derivative tensor $\nabla^d f$
- $F_1 \circ R_{d-1}$: tensor-vector $\nabla^d f \cdot \dot{x}$
- $F_d$:
  $$\left[\left[\left[\nabla^d f \cdot \dot{x}\right] \cdot \dot{x}\right] \cdots \right] \cdot \dot{x}$$
- The structural (and sparsity) properties of $T_f$ determines the optimal method.
- General compression and recovery using $F_1 \circ R_{d-1}$.

perfectly parallelizable