Extended Jacobian Chain Products

Dynamic Programming for Algorithmic Differentiation

Uwe Naumann

Informatik 12:
Software and Tools for Computational Engineering (STCE)

RWTH Aachen University

Contents

Objective and Learning Outcomes

Recall
- Chain Rule of Differential Calculus
- Dynamic Programming
- Algorithmic Differentiation

Extended Jacobian Chain Products
- Trace
- Case Study
- Implementation

Summary and Next Steps

Outline

Objective and Learning Outcomes

Recall

- Chain Rule of Differential Calculus
- Dynamic Programming
- Algorithmic Differentiation

Extended Jacobian Chain Products

- Trace
- Case Study
- Implementation

Summary and Next Steps

Software and Tools for Computational Engineering

RWTH AACHEN UNIVERSITY

Objective

- Introduction to optimization of Jacobian accumulation code by application of dynamic programming to extended Jacobian chain products.

Learning Outcomes

- You will understand
  - definition of trace of a differentiable computer program
  - construction of extended Jacobian chain products
- You will be able to
  - optimize Jacobian accumulation code by application of dynamic programming to extended Jacobian chain products

Outline

Objective and Learning Outcomes

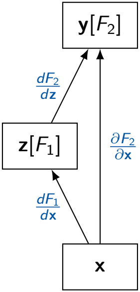
Recall
- Chain Rule of Differential Calculus
- Dynamic Programming
- Algorithmic Differentiation

Extended Jacobian Chain Products
- Trace
- Case Study
- Implementation

Summary and Next Steps

Recall

Chain Rule of Differential Calculus



Let $\mathbf{y} = F(\mathbf{x}) : \boldsymbol{R}^n \rightarrow \boldsymbol{R}^m$ be such that

$$\mathbf{y} = F(\mathbf{x}) = F_2(F_1(\mathbf{x}), \mathbf{x}) = F_2(\mathbf{z}, \mathbf{x})$$

with (continuously) differentiable $F_1 : \boldsymbol{R}^n \rightarrow \boldsymbol{R}^p$ and $F_2 : \boldsymbol{R}^p \times \boldsymbol{R}^n \rightarrow \boldsymbol{R}^m$.

Then F is continuously differentiable over $\boldsymbol{R}^n$ and

$$\frac{dF}{d\mathbf{x}}(\tilde{\mathbf{x}}) = \frac{dF_2}{d\mathbf{x}}(\tilde{\mathbf{z}}, \tilde{\mathbf{x}}) = \frac{dF_2}{d\mathbf{z}}(\tilde{\mathbf{z}}, \tilde{\mathbf{x}}) \cdot \frac{dF_1}{d\mathbf{x}}(\tilde{\mathbf{x}}) + \frac{\partial F_2}{\partial \mathbf{x}}(\tilde{\mathbf{z}}, \tilde{\mathbf{x}})$$

for all $\tilde{\mathbf{x}} \in \boldsymbol{R}^n$ and $\tilde{\mathbf{z}} = F_1(\tilde{\mathbf{x}})$.

Deeper nesting yields [sparse] matrix chain products.

The number of fused multiply-add (`fma`) operations required for the evaluation of a [sparse] matrix chain product

$$\prod_{\nu=p-1}^{0} A_\nu = A_{p-1} \cdot \ldots \cdot A_0 \quad \text{for } A_\nu = (a^\nu_{j,i})^{j=0,\ldots,m_\nu-1}_{i=0,\ldots,n_\nu-1} \in \boldsymbol{R}^{m_\nu \times n_\nu} \ .$$

can be reduced by dynamic programming

$$\mathtt{fma}_{k,i} = \begin{cases} 0 & k = i \\ \min_{i \le j < k} \left(\mathtt{fma}_{k,j+1} + \mathtt{fma}_{j,i} + \mathtt{fma}_{k,j,i} \right) & k > i \end{cases}$$

through tabulating the solutions $\mathtt{fma}_{k,i}$ of the subproblems $\prod_{\nu=k}^{i} A_\nu$ for $k - i = 0, \ldots, p$ and where $\mathtt{fma}_{k,j,i}$ is the cost of evaluating $A_{k,j} \cdot A_{j,i}$.

The same idea can be applied to [extended] Jacobian chain products arising from the chain rule of differential calculus.

Algorithmic Differentiation (AD) targets multivariate vector functions

$$F : \mathbb{R}^n \rightarrow \mathbb{R}^m : \mathbf{y} = F(\mathbf{x})$$

implemented as differentiable computer programs.

Such programs decompose into sequences of $q = p + m$ differentiable elemental functions φ_j evaluated as a [incremental] single assignment code

$$v_j = [v_j +]\, \varphi_j(v_k)_{k \prec j} \quad \text{for } j = 1, \ldots, q$$

and where $v_i = x_i$ for $i = 1 - n, \ldots, 0$, $[v_j = 0$ for $i = 1, \ldots, p,]$ $y_k = v_{p+k+1}$ for $k = 0, \ldots, m - 1$, and $k \prec j$ if v_k is an argument of φ_j.

A DAG $G = (V, E)$ is induced. Partial derivatives of the elemental functions wrt. their arguments are attached as labels to the corresponding edges.

Outline

Objective and Learning Outcomes

Recall
- Chain Rule of Differential Calculus
- Dynamic Programming
- Algorithmic Differentiation

Extended Jacobian Chain Products
- Trace
- Case Study
- Implementation

Summary and Next Steps

We consider the extended single assignment code

$$\mathbf{v}_0 = \begin{pmatrix} \mathbf{x} & \mathbf{0}_p & \mathbf{0}_m \end{pmatrix}^T$$
$$\mathbf{v}_j = \Phi_j(\mathbf{v}_{j-1}) \qquad j = 1, \ldots, q$$

with extended elemental functions Φ_j, $j = 1, \ldots, q$, whose k-th entry is defined as

$$[\Phi_j(\mathbf{v})]_k \equiv \begin{cases} v_j + \varphi_j(v_i)_{i \prec j} & \text{if } k = j \\ v_k & \text{otherwise} \end{cases}$$

yielding the trace $\Phi : \boldsymbol{R}^{n+q} \to \boldsymbol{R}^{n+q}$ as

$$\begin{pmatrix} \mathbf{x} & v_1 & \ldots & v_p & \mathbf{y} \end{pmatrix}^T = \mathbf{v}_q = \Phi(\mathbf{v}_0) = \Phi_q(\Phi_{q-1}(\ldots \Phi_1(\mathbf{v}_0) \ldots))$$

The trace computes the function value $\mathbf{y} = F(\mathbf{x})$ while keeping all intermediate values v_j, $j = 1, \ldots, p$. It induces a corresponding trace DAG.

By the chain rule, the Jacobian of the trace can be evaluated as the extended local Jacobian chain

$$\frac{dF}{d\mathbf{v}}(\mathbf{v}_0) = \frac{d\Phi_q}{d\mathbf{v}}(\mathbf{v}_{q-1}) \cdot \frac{d\Phi_{q-1}}{d\mathbf{v}}(\mathbf{v}_{q-2}) \cdot \ldots \cdot \frac{d\Phi_1}{d\mathbf{v}}(\mathbf{v}_0)$$

where entries of the extended local Jacobians are defined as

$$\left[\frac{d\Phi_j}{d\mathbf{v}}(\mathbf{v}_{j-1})\right]_{i,k} = \begin{cases} 1 & \text{if } k = i \\ \frac{d\varphi_i}{dv_k} & \text{if } k \prec i \\ 0 & \text{otherwise} . \end{cases}$$

Note that $\frac{dF}{d\mathbf{x}} = Q_m \cdot \frac{d\Phi}{d\mathbf{v}}(\mathbf{v}_0) \cdot P_n^T$ where

$Q_m \in \boldsymbol{R}^{m\times(n+q)}$ extracts the last m rows of $\frac{d\Phi}{d\mathbf{v}}(\mathbf{v}_0)$ when multiplied from the left, that is,

$$Q_m = \begin{pmatrix} 0_{m\times(n+p)} & I_m \end{pmatrix}$$

$P_n^T \in \boldsymbol{R}^{(n+q)\times n}$ extracts the first n columns of $\frac{d\Phi}{d\mathbf{v}}(\mathbf{v}_0)$ when multiplied from the right, that is,

$$P_n = \begin{pmatrix} I_n & 0_{n\times(p+m)} \end{pmatrix}$$

Elemental Extended Local Jacobians

Distribution of individual scalar local derivatives over elemental extended local Jacobians yields

$$\left[\frac{d\Phi_{j,i}}{d\mathbf{v}}\right]_{l,k} = \begin{cases} 1 & \text{if } l = k \\ \frac{d\varphi_j}{dv_i} & \text{if } l = j \text{ and } k = i \\ 0 & \text{otherwise} . \end{cases}$$

implying

$$\frac{d\Phi_j}{d\mathbf{v}} = \prod_{i \prec j} \frac{d\Phi_{j,i}}{d\mathbf{v}}$$

Note that the product of two elemental extended local Jacobians $\frac{d\Phi_{l,k}}{d\mathbf{v}}$ and $\frac{d\Phi_{j,i}}{d\mathbf{v}}$ is commutative if and only if $k \neq j$ ($i = l$ impossible due to topological order of scalar variables within the single assignment code).

Elemental Extended Local Jacobian Chain

$$\frac{d\Phi}{d\mathbf{v}} = \prod_{j=q}^{1} \prod_{i \prec j} \frac{d\Phi_{j,i}}{d\mathbf{v}}$$

Local commutativity yields a large number of variants including ...

Extended Local Tangent Chain

$$\frac{d\Phi}{d\mathbf{v}} = \prod_{j=q}^{1} \frac{d\Phi_j}{d\mathbf{v}}$$

Extended Local Adjoint Chain

$$\frac{d\Phi}{d\mathbf{v}} = \prod_{j=q-1}^{0} \frac{d\bar{\Phi}_j}{d\mathbf{v}} \quad \text{where} \quad \frac{d\bar{\Phi}_j}{d\mathbf{v}} = \prod_{i \succ j} \frac{d\Phi_{i,j}}{d\mathbf{v}}$$

Multiplication of the various extended extended local Jacobian chains with Q_m and P_n^T yields zero rows and columns the removal of which results in a pruned sparse rectangular extended local Jacobian chain.

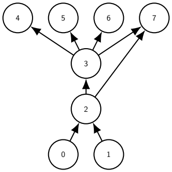
Extraction of the corresponding live section of the trace DAG amounts to keeping all edges/vertices lying on paths that connect x_i, $i = 0, \ldots, n-1$ with y_j, $j = 0, \ldots, m-1$, and discarding all others.

The pruned extended local Jacobian chain yields two dynamic programming formulations:

1. optimal bracketing of rectangular chain assuming dense factors;
2. optimal bracketing of sparse chain.

Case Study

Lion

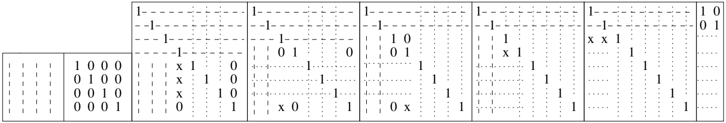


Let $A_j \equiv \frac{d\Phi_j}{d\mathbf{v}}$ and $A_{j,i} \equiv \frac{d\Phi_{j,i}}{d\mathbf{v}}$ in

$$\tilde{A}_6 \cdot \tilde{A}_5 \cdot \tilde{A}_4 \cdot A_3 \cdot A_2$$

with $\tilde{A}_6 = A_{6,3} \cdot A_{5,3} \cdot A_{4,3}$, $\tilde{A}_5 = A_{7,2}$, and $\tilde{A}_4 = A_{7,3}$.

Pruning yields



Application of dynamic programming to the pruned sparse chain yields the bracketing

$$\tilde{A}_6 \cdot ((\tilde{A}_5 \cdot (\tilde{A}_4 \cdot A_3)) \cdot A_2)$$

with a Jacobian accumulation cost of 11 `fma`.

As always, the challenge is for the special treatment of the combinatorics to pay off.

How to use the above to generate efficient Jacobian code?

Apply to static (run time invariant) parts of the code, i.e,

- build local DAGs
- derive pruned extended local Jacobian chain
- run dynamic programming algorithm
- use result to generate local Jacobian code
- run native optimizing compiler.

Combinatorial optimization of derivative code is useful in the context of source transformation.

Outline

Objective and Learning Outcomes

Recall
- Chain Rule of Differential Calculus
- Dynamic Programming
- Algorithmic Differentiation

Extended Jacobian Chain Products
- Trace
- Case Study
- Implementation

Summary and Next Steps

Summary

- Introduction to optimization of Jacobian accumulation code by application of dynamic programming to extended Jacobian chain products.
- Definition of trace of a differentiable computer program.
- Construction of extended Jacobian chain products.

Next Steps

- Practice derivation of extended Jacobian chain products.
- Continue the course to find out more ...