Optimized Jacobian Accumulation Techniques

UWE NAUMANN
Department of Computer Science
University of Hertfordshire
Hatfield, AL10 9AB
UNITED KINGDOM
U.1, Naumann@herts.ac.uk

Abstract: - Jacobian matrices can be accumulated using either the forward or reverse mode of Automatic Differentiation. Alternatively, derivative code can be generated to compute the Jacobian directly at the current argument. The minimisation of the corresponding number of arithmetic operations leads to a computationally hard combinatorial optimisation problem. A method for its approximate solution by dynamic programming will be discussed briefly. It results in a speedup of three and more for most problems.

Keywords: - Fast Jacobian Code, Combinatorial Optimisation, Dynamic Programming

1 Automatic Differentiation

Products of the form \((\mathbb{R}^{m \times n})\dot{X} = F' \dot{X}\) and \((\mathbb{R}^{m \times n})X = YF',\) where \(F' = \left(\frac{\partial f}{\partial x_j}(x_0)\right)_{j=1..n}\) is the Jacobian of a non-linear vector function

\[ F : \mathbb{R}^n \supseteq D \to \mathbb{R}^m : x \mapsto y = F(x) \quad (1) \]

evaluated at a given argument \(x_0\) are essential in many numerical computations.

**Assumption 1** \(F\) is given as a computer program which decomposes into a sequence of scalar elemental functions \((\mathbb{R} \supseteq) v_j = \varphi_j(v_i)_{i \in P_j}\) where \(j = 1..q\) and \(P_j \subseteq \{(1-n)..p\}\), \(p = q - m. \)

\(P_j\) is the set of indices of the arguments of \(\varphi_j\) and we denote its cardinality by \(|P_j|\). **Automatic Differentiation (AD)** [5], [3], [7] provides a fast and convenient way to compute derivatives with machine accuracy by simply exploiting the chain rule. AD should not be confused with the completely different approach of approximating derivatives numerically through finite difference quotients!

Within \(F\) we distinguish between three types of variables \(V = X \cup Z \cup Y\): the independent variables \(X \equiv \{v_{1-n},v_0\}\), the intermediate variables \(Z \equiv \{v_1,v_p\}\), and the dependent variables \(Y \equiv \{v_{p+1},v_q\}\). We set \(x_i \equiv v_{i-n}\), \(i = 1..n\), and \(y_j \equiv v_{p+j}\), \(j = 1..m\). The fact that \(v_j\) depends directly on \(v_i\) is represented by \(i \prec j\). The transitive closure of this relation will be denoted by \(\prec^*\). We expect the numbering \(\mathcal{I} : V \to \{(1-n)..q\}\) of the variables of \(F\) to be consistent, i.e. it must induce a topological order with respect to dependency as \(i \prec^* j \Rightarrow \mathcal{I}(v_i) < \mathcal{I}(v_j)\).
Since the differentiation of $F$ is based on the differentiability of its elemental functions the following will be assumed:

**Assumption 2** The $\varphi_j$ \((j = 1..q)\) have jointly continuous partial derivatives

$$c_{ji} \equiv \frac{\partial}{\partial x_i} \varphi_j(v_k)_{k \in P_j} \text{ for } i \in P_j$$  \hspace{1cm} (2)

on open neighborhoods $D_j \subset \mathbb{R}^n$ with $n_j \equiv |P_j|$ of their domain. \(\blacktriangleleft\)

Both the forward and the reverse modes of AD [9] are based on the application of the chain rule to $F$ in two opposite ways.

**Proposition 1 [Forward Mode of AD]** For $\mathbf{x} \in \mathbb{R}^n$ the product $\dot{\mathbf{y}} = F'\dot{\mathbf{x}}$ can be evaluated by $\dot{y}_j = \sum_{i \in P_j} c_{ji} \dot{v}_i$, $j = 1..q$, by the forward propagation of the intermediate derivatives $\mathbf{R} \ni \dot{v}_j = \left[\frac{\partial v_j}{\partial x_k}\right]_{k=1..n}^T \dot{x}$. \(\blacktriangleleft\)

**Proof:** The result follows from the chain rule

$$\frac{\partial y_j}{\partial x_{i+n}} = \frac{\partial y_j}{\partial v_i} = \sum_{k \in P_j} c_{jk} \frac{\partial v_k}{\partial v_i},$$  \hspace{1cm} (3)

where $j = (p+1)..<, i = (1-.n)..<$, and the $c_{jk}$ are defined by Eqn. (2). Explicit expressions for all entries of $F'$ are obtained by successively eliminating all terms containing intermediate variables. The procedure will be sketched by eliminating the dependence of $y_j, i \in (p+1)..<$, on some $v_{k_1}, k_1 \in P_{j_1}$, in Eqn. (3). Starting with the isolation of the term to be eliminated followed by the substitution of the expression corresponding to $\frac{\partial v_{k_1}}{\partial x_i}$, $i = 1..<$, by the chain rule, we get a new expression for calculating the derivative value $\dot{y}_{j_1-p}, j_1 = (p+1)..<$, which does not depend on $\dot{v}_{k_1}$, i.e.

$$\dot{y}_{j_1} = \sum_{k \in P_{j_1}} \tilde{c}_{j_1k} \dot{v}_k,$$  \hspace{1cm} (4)

where $\tilde{P}_{j_1} \equiv P_{k_1} \cup P_{j_1} \setminus \{k_1\}$ and

$$\tilde{c}_{j_1k} \equiv c_{j_1k} + c_{j_1k}c_{k_1}. \hspace{1cm} (5)$$

The repeated application of the above transformation to the derivatives $\dot{y}_j$, $j = 1..<$, of all dependent variables and their dependences on $\dot{v}_k$, $k = 1..<$, proves Prop. 1. \(\blacktriangleleft\) Prop. 2 is shown analogous.

**Proposition 2 [Reverse Mode of AD]** For $\mathbf{y} \in \mathbb{R}^m$ the product $\mathbf{x} = \mathbf{y}^T F'$ can be evaluated by $\mathbf{s}_j = \sum_{i \in S_j} c_{ji} \mathbf{s}_i$, $j = p.(1-.n)$, by the backward propagation of intermediate adjoints. $\mathbf{R} \ni \mathbf{s}_j = \mathbf{y}^T \left[\frac{\partial s_j}{\partial x_k}\right]_{k=1..<}$. \(\blacktriangleleft\)

The expression $\frac{\partial s_j}{\partial x_k}$ should be understood as the sensitivity of $y_k$ with respect to $v_j$.

## 2 EJA Problem

While the forward and reverse modes of AD are merely two special choices the chain rule can be applied to $F$ in any arbitrary order. These various orderings lead to up to round-off identical numerical results but may have drastically varying operations counts for the accumulation of $F'$. The general task of evaluating $F'$ efficiently - the Efficient Jacobian Accumulation problem - is conjectured to be NP-hard [6]. There are a few works by various authors that motivate a closer look at this topic, e.g. [8], [4]. More recent research lead to several so far unknown results partly presented in [10].

AD can be regarded as a program transformation technique that takes $F$ and returns a program for computing $\mathbf{Y} = F'(\mathbf{x}) \mathbf{X}$ (forward mode) or $\mathbf{X} = \mathbf{Y} F'(\mathbf{x})$ (reverse mode). $F' = F'(\mathbf{x})$ can be computed by setting $\mathbf{X}$ or $\mathbf{Y}$ equal to the identity matrix, respectively. Jacobian accumulation is considered to be a method
transforming \( F \) into a program which evaluates \( F' \) at the given argument \( x_0 \) directly.

Depending on the order in which the chain rule will be applied to \( F \) the accumulation of \( F' \) will involve the evaluation of varying numbers of maf’s (multiply-add-fused) defined by Eqn. (5).

**Assumption 3** The evaluation of a maf takes the same time as a single multiplication. The computational cost is equivalent to the number of maf’s.

Since there are various methods \( M \) for accumulating \( F' \) let \( MAF_F(M) \) denote the computational cost of accumulating \( F' \) by applying some method \( M \) to \( F \). However, the number of arithmetic operations performed is not the only factor influencing the runtime of an algorithm.

**Assumption 4** The size \( n+q \) of \( F \) is such that all partial derivatives defined by Eqn. (2) can be held in the main memory. There are no significant differences in the time required to access different parts of the memory.

To find a method \( M \) for accumulating \( F' \) using a minimal number of maf’s for arbitrary \( F \) the following combinatorial optimization problem has to be solved:

\[
\text{Find } M \text{ s.t. } (MAF_F(M) \rightarrow \text{MIN}) \forall F
\]

where \( M \) is some method for accumulating Jacobians.

The above represents a more formal formulation of the EJA problem. Several approaches based on the elimination of vertices, edges and faces in computational graphs have been developed. Local heuristics as well as simulated annealing and dynamic programming techniques lead to a remarkable speedup of the optimized Jacobian code compared to established and widely used methods. First numerical results have been presented in [8] and [10]. The problem is being investigated in ongoing research.

### 3 Matrix chains

In this paper we will present an approach to the approximate solution of the EJA problem based on the principles of chained matrix products and dynamic programming.

**Definition 1** Let \( C_j = C_j' + I, \tilde{C}_j = \tilde{C}_j' + I \), such that \( C_j, \tilde{C}_j \in \mathbb{R}^{(n+q) \times (n+q)} \) for \( j, i = (1 - n) \cdot q \) and where

\[
C'_j = \begin{cases} 
c_{ji} & \text{if } i < j \\
0 & \text{otherwise}
\end{cases}, \quad \tilde{C}'_j = \begin{cases} 
c_{ij} & \text{if } j < i \\
0 & \text{otherwise}
\end{cases}
\]

while \( I \) denotes the identity matrix in \( \mathbb{R}^{(n+q) \times (n+q)} \).

\( C_j \) combines the partial derivatives of \( v_j \) w.r.t. all its predecessors. Analogous, \( \tilde{C}_j \) consists of the partial derivatives of all successors of \( v_j \) w.r.t. itself. Let \( P_n \) and \( Q_m \) be matrices that project a given vector of length \( n+q \) to its first \( n \) and its last \( m \) components, respectively. As a direct consequence of the chain rule we get the following result:

**Proposition 3**

\[
P' = Q_m \left( \prod_{j=1}^{q} C_j \right) P_n^T = Q_m \left( \prod_{j=1-n}^{p} \tilde{C}_j \right) P_n^T.
\]

The formal proof is a straight-forward, but rather technical, argument based on Prop. 1 and Prop. 2, which will not be presented here. The product \( \prod_{j=1}^{q} C_j \) is referred to as the tangent chain. \( \prod_{j=1-n}^{p} \tilde{C}_j \) is called the adjoint chain.
4 Solving the EJA Problem

The associativity of matrix multiplication in both the tangent and the adjoint chains may be exploited in order to solve the EJA problem approximately. W.l.o.g. let us concentrate on the tangent chain. The objective is to determine an order in which the factors are multiplied such that the number of maf’s becomes minimal. This combinatorial optimization problem, which is closely related to the EJA problem, will be referred to as the Tangent Chain Optimization (TCO) problem. As a chained matrix multiplication problem [1] the TCO problem has two key ingredients that makes it suitable for dynamic programming [1]: Optimal substructure and overlapping subproblems. A problem exhibits optimal substructure if an optimal solution to the problem is built on optimal solutions to subproblems. Furthermore, the space of these subproblems must be small in the sense that a recursive algorithm for the problem repeatedly solves the same subproblems, rather than always generating new instances. Ideally, the number of distinct subproblems grows polynomially with the input size of the composite problem. Dynamic programming algorithms take advantage of overlapping subproblems by solving each subproblem once. The results are stored in a table where they can be looked up when needed, using constant time.

Let \( m_{j,i} \) be the minimal number of maf’s required for computing the products \( C_{j,i} = C_j \ldots C_i \) for \( q \geq j \geq i \geq 1 \) and \( m_{jki} \) is to denote the cost of multiplying \( C_{j,k+1} \) by \( C_{k,i} \). The key difference between the classical chained matrix problem and the TCO problem is that the cost increments \( m_{jki} \) to merge \( C_{i,(k+1)} \) and \( C_{k,j} \) cannot be calculated cheaply a priori in the second case. The sparsity pattern of all intermediate products have to be computed explicitly which leads to an increased complexity \( O((n + q)^3) \) of the following dynamic programming algorithm:

\[
m_{j,i} = \begin{cases} 0, & \text{if } i = j; \\ \min_{i \leq k < j} \{m_{k,i} + m_{j,k+1} + m_{jki}\}, & \text{if } j > i. \end{cases}
\]

5 Numerical Results

In the table below we have compared the computational costs of accumulating the Jacobian using a method due to Newsam and Ramsdell [11] (one of the best Jacobian accumulation methods currently available) with the results obtained by solving the TCO problem. We consider three MINPACK test problems [2]: WAT - Watson function \((\mathbb{R}^7 \rightarrow \mathbb{R}^7)\); DIE - Discrete integral equation function \((\mathbb{R}^{20} \rightarrow \mathbb{R}^{20})\); GDF - Gaussian data fitting problem \((\mathbb{R}^{11} \rightarrow \mathbb{R}^{65})\).

<table>
<thead>
<tr>
<th>p</th>
<th>NR</th>
<th>DP</th>
<th>NR / DP</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAT</td>
<td>1683</td>
<td>11830</td>
<td>4240</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2.8</td>
</tr>
<tr>
<td>DIE</td>
<td>2499</td>
<td>50380</td>
<td>1659</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>30.3</td>
</tr>
<tr>
<td>GDF</td>
<td>1625</td>
<td>18590</td>
<td>1430</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>13.0</td>
</tr>
</tbody>
</table>

Remarkable savings in the operations count can be achieved already by fully exploiting the structural sparsity of a problem. In [10] we have compared the run times of an optimized derivative code and a state-of-the-art AD based code. The objective was to show that the theoretical improvements in the operations count could be "translated" into run time savings. Promising a reduction of the operations count by a factor of 4 the resulting optimized derivative code ran 3.5 times faster than the original version.
6 Further Steps

The TCO problem restricts the possible application sequences of the chain rule such that a polynomial dynamic programming algorithm can be constructed for its solution.

**Proposition 4** The solution of the TCO problem is in general not the solution of the EJA problem.

**Proof:** We will prove the above by presenting an example. Consider the following specification of a non-linear vector function $F$:

$$v_1 = \varphi_1(v_{-1}, v_0); \quad v_2 = \varphi_2(v_1);$$
$$v_3 = \varphi_3(v_1, v_2); \quad v_k = \varphi_k(v_2) \quad (k = 4..7)$$

where $x_i = v_{i-2}$ for $i = 1, 2$ and $y_j = v_{j+2}$, $j = 1..5$. The corresponding tangent chain is given by $\prod_{i=1}^7 C_i = C_7 C_6 .. C_1$. Assume that maf’s involving diagonal elements of the elementary extended Jacobians, which are all equal to one, are not counted. Using the notation $C_{i,j} = \prod_{k=i}^j C_k$ it is easy to check that $\text{MAF}_{F'}[C_{4,7}((C_3 C_2) C_1)] = 13$ while, for example, $\text{MAF}_{F'}[C_{3,7}(C_2 C_1)] = 14$. In [10] we have shown that the minimal cost of accumulating the Jacobian of $F$ is, in fact, 13. However, the success of the dynamic programming algorithm depends on the numbering of the intermediate variables which we have chosen "incidently" in a suitable way. Suppose that we switch two of the indices such that $v_3 = \varphi_3(v_2)$ and $v_4 = \varphi_3(v_1, v_2)$ which would still represent a consistent set of the intermediate variables of $F$ in the sense of Sec. 1. We leave it to the reader to verify that the evaluation of the resulting chained matrix product $C_7 C_6 C_5 C_3 C_4 C_2 C_1$ would involve, at least, 14 maf’s.

The above example gives us the opportunity to have a brief look at what we actually want to get as a result - Jacobian code. The reverse mode of AD could be applied to generate the following adjoint code:

$$\bar{v}_2 = \sum_{i=7}^3 c_{i,2} \bar{v}_i; \quad \bar{v}_1 = \sum_{i=3}^2 c_{i,1} \bar{v}_i;$$
$$\bar{v}_0 = c_{1,0} \bar{v}_1; \quad \bar{v}_{-1} = c_{1,-1} \bar{v}_1.$$  

According to Sec. 1 $F'$ can be accumulated by propagating the five Cartesian basis vectors in $\mathbb{R}^5$ backwards, i.e. five runs of the adjoint code are required with $\bar{y} = (v_k)_{k=3..7} = e_i$, $i = 1..5$. Even if multiplications by 0 are not counted - which, in general, is rather hard to implement - this would still result in 21 maf’s. Analogous, the forward mode would deliver a tangent linear code two runs of which would be required to accumulate $F'$ performing 16 maf’s. The optimal Jacobian code corresponding to the parenthesisation $C_{4,7}((C_3 C_2) C_1)$ in the proof of Prop. 4 delivers $F'$ after performing only 13 maf’s:

$$c_{3,1} = c_{3,1} + c_{3,2} c_{2,1};$$
$$c_{2,i} = c_{2,i} c_{1,i} \quad (i = -1, 0);$$
$$F'_{k-1} = c_{k,2} c_{2,-1} \quad (k = 4..7);$$
$$F'_{k,0} = c_{k,2} c_{2,0} \quad (k = 4..7);$$
$$F'_{3,i} = c_{3,1} c_{1,i} \quad (i = -1, 0).$$

7 Conclusion, Outlook

The TCO problem is a special case of the Optimal Edge Elimination (OEE) problem in linearized computational graphs [10]. There are two ways to eliminate an edge $(i, j)$ corresponding to the two matrix products $C_j C_i$ (back elimination) $C_j C_i$ (front elimination). Graphically the back [front] elimination of $(i, j)$ is equivalent to connecting all predecessors of $v_i$ with $v_j$.
\[ v_i \text{ with all successors of } v_j \] followed by the removal of \((i, j)\). The number of \texttt{map}'s involved in the back [front] elimination of an edge is equal to the number of predecessors of its source [successors of its target]. Obviously, the successive back elimination [front elimination] of all edges emanating from [leading into] a vertex \(v_j\) is equivalent to the elimination of \(v_j\) itself. Local heuristics and simulated annealing algorithms are being developed to solve the edge elimination problem approximately. It can be shown that an optimal solution of the OEE problem does in general not solve the EJA problem.

The generation of efficient Jacobian code based on optimized elimination sequences may lead to a remarkable speedup of available simulation and optimization algorithms. So far, there is no software tool for AD realizing this. In cooperation with several academic and industrial partners we are about to launch a European project to overcome this deficit.

References


