Randomized Heuristics for Exploiting Jacobian Scarcity

Andrew Lyons$^{1,2}$ Ilya Safro$^2$

$^1$University of Chicago
$^2$Argonne National Laboratory

Dagstuhl Seminar on Combinatorial Scientific Computing
February 3, 2009
Outline

Introduction and Motivation
  Linearization
  Vector Propagation
  Preaccumulation

Jacobian Scarcity
  Structural Properties of Jacobians
  Randomized Algorithms
  Results
  Rerouting and Normalization

Future Work
  Connections to Optimal Jacobian Accumulation
  Leveraging Face Elimination
Given program for \( y = F(x) : \mathbb{R}^n \rightarrow \mathbb{R}^m \)
Context

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

Want a program $F^+(\mathbf{x})$ that computes $F(\mathbf{x})$ plus a collection of $p$ Jacobian-vector products

$$F'(\mathbf{x})\dot{\mathbf{x}}^1, F'(\mathbf{x})\dot{\mathbf{x}}^2, \ldots, F'(\mathbf{x})\dot{\mathbf{x}}^p$$
Context

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

Want a program $F^+(\mathbf{x})$ that computes $F(\mathbf{x})$ plus a collection of $p$ Jacobian-vector products

$$F'(\mathbf{x})\dot{\mathbf{x}}^1, F'(\mathbf{x})\dot{\mathbf{x}}^2, \ldots, F'(\mathbf{x})\dot{\mathbf{x}}^p$$

or a collection of $p$ Jacobian-transpose-vector products

$$F'(\mathbf{x})^T \bar{\mathbf{y}}^1, F'(\mathbf{x})^T \bar{\mathbf{y}}^2, \ldots, F'(\mathbf{x})^T \bar{\mathbf{y}}^p$$

Assume $p \gg \max\{n, m\}$ (we want a lot of them) As needed in Newton Krylov methods, etc.
Given program for $y = F(x) : \mathbb{R}^n \rightarrow \mathbb{R}^m$

Want a program $F^+(x)$ that computes $F(x)$ plus a collection of $p$ Jacobian-vector products

$$F'(x)\dot{x}^1, F'(x)\dot{x}^2, \ldots, F'(x)\dot{x}^p$$

or a collection of $p$ Jacobian-transpose-vector products

$$F'(x)^T\bar{y}^1, F'(x)^T\bar{y}^2, \ldots, F'(x)^T\bar{y}^p$$

Assume $p \gg \max\{n, m\}$ (we want a lot of them)
As needed in Newton Krylov methods, etc.
Evaluation Procedures

\[ y_1 = x_1 \times x_1 \times x_2, \quad y_2 = \sin(x_1 \times x_1 \times x_2) \]

\[
\begin{align*}
v_{-1} & = x_1 \\
v_0 & = x_2 \\
v_1 & = v_{-1} \times v_0
\end{align*}
\]

\[
\begin{align*}
v_2 & = v_{-1} \times v_1 \\
v_3 & = v_2 + 1 \\
v_4 & = \sin(v_2)
\end{align*}
\]

\[
\begin{align*}
y_1 & = v_3 \\
y_2 & = v_4
\end{align*}
\]
Computational Graphs and Evaluation Procedures

\[ y_1 = x_1 \times x_1 \times x_2, \quad y_2 = \sin(x_1 \times x_1 \times x_2) \]

\[ v_{-1} = x_1 \]
\[ v_0 = x_2 \]
\[ v_1 = v_{-1} \times v_0 \]

\[ v_2 = v_{-1} \times v_1 \]
\[ v_3 = v_2 + 1 \]
\[ v_4 = \sin(v_2) \]

\[ y_1 = v_3 \]
\[ y_2 = v_4 \]
Computational Graphs and Linearization

\[ y_1 = x_1 \times x_1 \times x_2, \quad y_2 = \sin(x_1 \times x_1 \times x_2) \]

\[
\begin{align*}
  v_{-1} &= x_1 \\
  v_0 &= x_2 \\
  v_1 &= v_{-1} \times v_0 \\
  c_{1-1} &= v_0 \\
  c_{10} &= v_1 \\
  v_2 &= v_{-1} \times v_1 \\
  c_{2-1} &= v_1 \\
  c_{21} &= v_{-1} \\
  v_3 &= v_2 + 1 \\
  c_{32} &= 1 \\
  v_4 &= \sin(v_2) \\
  c_{42} &= \cos(v_2) \\
  y_1 &= v_3 \\
  y_2 &= v_4
\end{align*}
\]
Forward **Propagation** of Vectors

Associate a derivative vector $\mathbf{\dot{v}}_j \in \mathbb{R}^p$ with each variable $v_j$, propagate through $G$ by

$$
\mathbf{\dot{v}}_j = \sum_{i \prec j} c_{ji} \mathbf{\dot{v}}_i \quad \text{(BLAS level 1 axpy operation)}
$$

![Diagram of the propagation process]

Generated propagation code:

$$
\mathbf{\dot{v}}_j = c_{ji} \mathbf{\dot{v}}_i \ldots
\mathbf{\dot{v}}_j = c_{ji'} \mathbf{\dot{v}}_{i'}
$$
Forward **Propagation** of Vectors

Associate a derivative vector \( \dot{\mathbf{v}}_j \in \mathbb{R}^p \) with each variable \( v_j \), propagate through \( G \) by

\[
\dot{\mathbf{v}}_j = \sum_{i \prec j} c_{ji} \dot{\mathbf{v}}_i \quad \text{(BLAS level 1 axpy operation)}
\]

**Generated propagation code:**

\[
\begin{align*}
\dot{v}_j &= c_{ji} \cdot \dot{v}_i \\
& \vdots \\
\dot{v}_j &= + c_{ji'} \cdot \dot{v}_{i'}
\end{align*}
\]
Forward Propagation of Vectors

\[
\begin{align*}
\dot{v}_{-1} &= \dot{x}_1 \\
\dot{v}_0 &= \dot{x}_2 \\
\end{align*}
\]
Forward Propagation of Vectors

\[ v_{-1} = x_1 \]
\[ v_0 = x_2 \]
\[ v_1 = c_{1-1} \cdot v_{-1} \]
\[ v_1 += c_{10} \cdot v_0 \]

\[ v_3 \]
\[ v_4 \]
Forward Propagation of Vectors

\[ \dot{v}_1 = c_{1-1} \cdot \dot{v}_1 \]
\[ \dot{v}_1 += c_{10} \cdot \dot{v}_0 \]
\[ \dot{v}_2 = c_{2-1} \cdot \dot{v}_{-1} \]
\[ \dot{v}_2 += c_{21} \cdot \dot{v}_1 \]

\[ \dot{v}_0 = \dot{x}_2 \]

\[ \dot{v}_{-1} = \dot{x}_1 \]
Forward Propagation of Vectors

\[ \dot{v}_{-1} = \dot{x}_1 \]
\[ \dot{v}_0 = \dot{x}_2 \]

\[ \dot{v}_1 = c_{1-1} \times \dot{v}_{-1} \]
\[ \dot{v}_1 += c_{10} \times \dot{v}_0 \]
\[ \dot{v}_2 = c_{2-1} \times \dot{v}_{-1} \]
\[ \dot{v}_2 += c_{21} \times \dot{v}_1 \]
\[ \dot{v}_3 = c_{32} \times \dot{v}_2 = \dot{v}_2 \]
Forward Propagation of Vectors

\[ \begin{align*}
\dot{v}_0 &= \dot{x}_2 \\
\dot{v}_1 &= c_{1-1} * \dot{v}_-1 \\
\dot{v}_1 &= c_{10} * \dot{v}_0 \\
\dot{v}_2 &= c_{2-1} * \dot{v}_-1 \\
\dot{v}_2 &= c_{21} * \dot{v}_1 \\
\dot{v}_3 &= c_{32} * \dot{v}_2 = \dot{v}_2 \\
\dot{v}_4 &= c_{42} * \dot{v}_2
\end{align*} \]
**Forward Propagation of Vectors**

1. \( \dot{v}_{-1} = \dot{x}_1 \)
2. \( \dot{v}_0 = \dot{x}_2 \)
3. \( \dot{v}_1 = c_{1-1} \cdot \dot{v}_{-1} \)
4. \( \dot{v}_1 += c_{10} \cdot \dot{v}_0 \)
5. \( \dot{v}_2 = c_{2-1} \cdot \dot{v}_{-1} \)
6. \( \dot{v}_2 += c_{21} \cdot \dot{v}_1 \)
7. \( \dot{v}_3 = c_{32} \cdot \dot{v}_2 \)
8. \( \dot{v}_4 = c_{42} \cdot \dot{v}_2 \)

\[ \begin{align*}
\dot{y}_1 &= \dot{v}_3 \\
\dot{y}_2 &= \dot{v}_4
\end{align*} \]

5 \( \rho \) multiplies
Reverse Vector Propagation

Propagates vectors $\bar{y}^1, \ldots, \bar{y}^p$ backwards

Works symmetrically
(same mult. cost, possibly different number of adds)

Yields Jacobian-transpose-vector products

$$F'(x)^T \bar{y}^1, F'(x)^T \bar{y}^2, \ldots, F'(x)^T \bar{y}^p$$
Preaccumulation

Cost is proportional to the number of nonunit edges ⇒ transform the graph!

Baur’s formula (from chain rule) yields the entries of $F'(x)$

$$\frac{\partial y_j}{\partial x_i} = \sum_{P \in P_{x_i}^{y_j}} \prod_{(k, \ell) \in P} c_{\ell k}$$
Preaccumulation

Cost is proportional to the number of nonunit edges ⇒ transform the graph!

Baur’s formula (from chain rule) yields the entries of $F'(x)$

$$\frac{\partial y_j}{\partial x_i} = \sum_{P \in \mathcal{P}_{x_i}^{y_j}} \prod_{(k, \ell) \in P} c_{\ell k}$$

Preaccumulation applies transformations $G \rightarrow G'$
Afterwards, Baur’s formula still expresses the entries of $J$ (we can still propagate vectors through it)
Preaccumulation

Cost is proportional to the number of nonunit edges ⇒ transform the graph!

Baur’s formula (from chain rule) yields the entries of $F'(x)$

$$\frac{\partial y_j}{\partial x_i} = \sum_{P \in P_{x_i}^{y_j}} \prod_{(k, \ell) \in P} c_{\ell k}$$

Preaccumulation applies transformations $G \rightarrow G'$
Afterwards, Baur’s formula still expresses the entries of $J$ (we can still propagate vectors through it)

Complete preaccumulation results in a bipartite graph, whose edges correspond to the nonzero entries of $F'(x)$
(In general, complete preaccumulation with minimal ops (OJA) is NP-hard)
Baur’s Formula

\[ \frac{\partial y_j}{\partial x_i} = \sum_{P \in \mathcal{P}_{x_i}^{y_j}} \prod_{(k, \ell) \in P} c_{\ell k} \]

\[ F'(x) = \begin{bmatrix} c_{2-1}c_{32} + c_{1-1}c_{21}c_{32} & c_{10}c_{21}c_{32} \\ c_{2-1}c_{42} + c_{1-1}c_{21}c_{42} & c_{10}c_{21}c_{42} \end{bmatrix} \]

\[ F'(x) = \begin{bmatrix} c_{2-1} + c_{1-1}c_{21} & c_{10}c_{21} \\ c_{2-1}c_{42} + c_{1-1}c_{21}c_{42} & c_{10}c_{21}c_{42} \end{bmatrix} \]
Generated preaccumulation code:

\[ c_{ki} += c_{ji} \cdot c_{kj} \]
\[ c_{k'i} = c_{ji} \cdot c_{k'j} \]
Front Edge Elimination

Generated preaccumulation code:

\[
\begin{align*}
C_{ki} & \;+\; C_{ji} \;\times\; C_{kj} \\
\vdots & \\
C_{k'i} & \;=\; C_{ji} \;\times\; C_{k'j}
\end{align*}
\]
Back Edge Elimination

\[ c_{ki} \]  
\[ c_{kj} \]  
\[ c_{jj'} \]  
\[ c_{ji} \]  
\[ c_{ki} += c_{ji} \cdot c_{kj} \]
Back Edge Elimination

\[
C_{ki} + C_{ji} \times C_{kj} \\
\vdots \\
C_{ki'} = C_{ji'} \times C_{kj}
\]
Preaccumulation

(Code for $F$, linearization)

\[
\begin{align*}
\dot{v}_3 &= c_{32} \cdot \dot{v}_{-1} + c_{30} \cdot \dot{v}_0 \\
\dot{v}_4 &= c_{42} \cdot \dot{v}_{-1} + c_{40} \cdot \dot{v}_0
\end{align*}
\]
Preaccumulation

(Code for $F$, linearization)

\[
\begin{align*}
\mathbf{c}_{2-1} &= \mathbf{c}_{1-1} \times \mathbf{c}_{21} \\
\mathbf{c}_{20} &= \mathbf{c}_{10} \times \mathbf{c}_{21}
\end{align*}
\]
Preaccumulation

(Code for $F$, linearization)

\[
\begin{align*}
    c_{2-1} &= c_{1-1} \times c_{21} \\
    c_{20} &= c_{10} \times c_{21}
\end{align*}
\]
Preaccumulation

(Code for $F$, linearization)

\[
\begin{align*}
    c_{2-1} &= c_{1-1} \times c_{21} \\
    c_{20} &= c_{10} \times c_{21} \\
    c_{3-1} &= c_{2-1} \times c_{32} \\
    c_{4-1} &= c_{2-1} \times c_{42}
\end{align*}
\]
Preaccumulation

(Code for $F$, linearization)

\[
\begin{align*}
    c_{2-1} &= c_{1-1} \times c_{21} \\
    c_{20} &= c_{10} \times c_{21} \\
    c_{3-1} &= c_{2-1} \times c_{32} \\
    c_{4-1} &= c_{2-1} \times c_{42}
\end{align*}
\]
(Code for $F$, linearization):

\begin{align*}
    c_{2-1} &= c_{1-1} \times c_{21} \\
    c_{20} &= c_{10} \times c_{21} \\
    c_{3-1} &= c_{2-1} \times c_{32} \\
    c_{4-1} &= c_{2-1} \times c_{42} \\
    c_{30} &= c_{20} \times c_{32} \\
    c_{40} &= c_{20} \times c_{42}
\end{align*}
Preaccumulation

(Code for $F$, linearization)

\[
\begin{align*}
C_{2-1} &= C_{1-1} \times C_{21} \\
C_{20} &= C_{10} \times C_{21} \\
C_{3-1} &= C_{2-1} \times C_{32} \\
C_{4-1} &= C_{2-1} \times C_{42} \\
C_{30} &= C_{20} \times C_{32} \\
C_{40} &= C_{20} \times C_{42}
\end{align*}
\]
Preaccumulation

(Code for $F$, linearization)

\[
\begin{align*}
C_{2-1} &= C_{1-1} \times C_{21} \\
C_{20} &= C_{10} \times C_{21} \\
C_{3-1} &= C_{2-1} \times C_{32} \\
C_{4-1} &= C_{2-1} \times C_{42} \\
C_{30} &= C_{20} \times C_{32} \\
C_{40} &= C_{20} \times C_{42} \\
\end{align*}
\]

4 mults

\[
\begin{align*}
\dot{v}_3 &= C_{3-1} \times \dot{v}_1 \\
\dot{v}_3 &= C_{30} \times \dot{v}_0 \\
\dot{v}_4 &= C_{4-1} \times \dot{v}_1 \\
\dot{v}_4 &= C_{40} \times \dot{v}_0 \\
\end{align*}
\]

4p mults
Costs

Fixed costs: Evaluation of $F$ and linearization
Costs

Fixed costs: Evaluation of $F$ and linearization

Propagation: $5p$ multiplications
Costs

Fixed costs: Evaluation of $F$ and linearization

Propagation: $5p$ multiplications

Complete Preaccumulation + Propagation: $4 + 4p$ multiplications
(Code for $F$, linearization)

\[ 
\begin{align*}
 c_{2-1} &= c_{1-1} * c_{21} & 2 \text{ mults} \\
 c_{20} &= c_{10} * c_{21}
\end{align*}
\]

\[ 
\begin{align*}
 \dot{v}_2 &= c_{2-1} * \dot{v}_{-1} \\
 \dot{v}_2 &=+ c_{20} * \dot{v}_0 \\
 \dot{v}_3 &= c_{32} * \dot{v}_2 = \dot{v}_2 \\
 \dot{v}_4 &= c_{42} * \dot{v}_2
\end{align*}
\]
Costs

Fixed costs: Evaluation of $F$ and linearization

**Propagation:** $5p$ multiplications

**Complete Preaccumulation + Propagation:** $4 + 4p$ multiplications

**Partial Preaccumulation + Propagation:** $2 + 3p$ multiplications

⇒ Assume $p$ is large, so ignore preaccumulation cost and focus on propagation cost.
Outline

Introduction and Motivation
  Linearization
  Vector Propagation
  Preaccumulation

Jacobian Scarcity
  Structural Properties of Jacobians
  Randomized Algorithms
  Results
  Rerouting and Normalization

Future Work
  Connections to Optimal Jacobian Accumulation
  Leveraging Face Elimination
Jacobian Scarcity

Example Jacobian happens to be dense, but some structure is lost when $F'(x)$ is accumulated to a matrix. For example, the Jacobian is low rank (\# of vertex-disjoint paths).

$\Rightarrow$ Jacobian **scarcity** (Griewank)
Example Jacobian happens to be dense, but some structure is lost when $F'(x)$ is accumulated to a matrix. For example, the Jacobian is low rank ($\#$ of vertex-disjoint paths).

⇒ Jacobian **scarcity** (Griewank)

Scarcity is a kind of deficiency, approximated by the number of nonunit edges in $G$.
Graph transformations that *don't increase* the number of nonunit edges are said to be **scarcity preserving**.
Jacobian Scarcity

Example Jacobian happens to be dense, but some structure is lost when $F'(x)$ is accumulated to a matrix. For example, the Jacobian is low rank (# of vertex-disjoint paths).

⇒ Jacobian **scarcity** (Griewank)

Scarcity is a kind of deficiency, approximated by the number of nonunit edges in $G$

Graph transformations that *don’t increase* the number of nonunit edges are said to be **scarcity preserving**.

⇒ Exploiting Scarcity: finding minimal representation of $G(F'(x))$
Jacobian Scarcity

Example Jacobian happens to be dense, but some structure is lost when $F'(x)$ is accumulated to a matrix. For example, the Jacobian is low rank (# of vertex-disjoint paths).

⇒ Jacobian **scarcity** (Griewank)

Scarcity is a kind of deficiency, approximated by the number of nonunit edges in $G$
Graph transformations that *don’t increase* the number of nonunit edges are said to be **scarcity preserving**.

⇒ Exploiting Scarcity: finding minimal representation of $G(F'(x))$

Greedy Algorithm - Lyons & Utke (AD2008)
Scarcity in Practice
Metagraph $M(G)$

Transitions in $M(G) \leftrightarrow$ change in number of nonunit edges $|E^+|$
Randomized Traversal of the Metagraph

Simple random walk: yields up to 10% improvement over greedy algorithm
Randomized Traversal of the Metagraph

Simple random walk: yields up to 10% improvement over greedy algorithm

...but it takes a LONG TIME (weeks?)
Randomized Traversal of the Metagraph

Simple random walk: yields up to 10% improvement over greedy algorithm

...but it takes a LONG TIME (weeks?)

Nice proof of concept, but can we do better? (I hope so!)
Simulated Annealing vs. Metropolis

Assign each transition (including backwards) in the metagraph a probability based on the change in energy $\delta E$

$$Pr = e^{-\frac{\delta E}{kT}}$$

where $T$ is a temperature and $k$ is a constant.

- Simulated Annealing: gradual heating/cooling scheme up to 20% improvement
- Metropolis: fixed $T$ – up to 35%+ improvement
Simulated Annealing vs. Metropolis

“Surprisingly enough, many problems cannot be solved more efficiently by SA than by the Metropolis.”

“Surprisingly enough, many problems cannot be solved more efficiently by SA than by the Metropolis.”


Our conclusion: Our best results were obtained with a fixed temperature scheme (Metropolis). Variations in temperature don’t appear to help much (plus adds additional parameter).
Simulated Annealing vs. Metropolis

“Surprisingly enough, many problems cannot be solved more efficiently by SA than by the Metropolis.”


Our conclusion: Our best results were obtained with a fixed temperature scheme (Metropolis). Variations in temperature don’t appear to help much (plus adds additional parameter).

▶ Bound the running time scheme by $100|V_{orig}|$ transitions
▶ Small number (5 or so) of restarts
▶ Runs in minutes on a laptop (not performed at runtime)
Results

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>259$p$</td>
<td>119 + 231$p$</td>
<td>146 + 222$p$</td>
</tr>
<tr>
<td>II</td>
<td>108$p$</td>
<td>34 + 93$p$</td>
<td>129 + 83$p$</td>
</tr>
<tr>
<td>III</td>
<td>241$p$</td>
<td>372 + 185$p$</td>
<td>780 + 140$p$</td>
</tr>
</tbody>
</table>

(Metric is multiplications performed when propagating $p$ vectors.)
Edge Prerouting

\[ i \quad j \quad k \quad \ell \quad \ell' \]

\[ c_{ji} = t, \quad c_{\ell i} = c_{\ell j} \]

\[ k \rightarrow \ell \rightarrow \ell' \rightarrow j \rightarrow i \]

\[ i \rightarrow j \rightarrow \ell \rightarrow \ell' \]

\[ i \rightarrow j \rightarrow k \rightarrow \ell \]

\[ i \rightarrow j \rightarrow k \rightarrow \ell' \]

\[ i \rightarrow j \rightarrow \ell \rightarrow \ell' \]
\[ t = \frac{c_{ki}}{c_{kj}} \]
\[ c_{ji} += t \]
\[ c_{\ell i} -= c_{\ell j} \times t \]
\[ \vdots \]
\[ c_{\ell' i} -= c_{\ell' i} \times t \]
Edge **Normalization** Forward

\[\ell \quad \ldots \quad \ell' \quad k \quad j \quad \ldots \quad j'\]

\[\frac{c_{kj}}{c_{ki}} = 1\]
Edge Normalization Forward

\[
\begin{align*}
&\ell \rightarrow i \rightarrow j \rightarrow \cdots \rightarrow j' \rightarrow \ell' \\
&\ell' \rightarrow \ell \rightarrow i \rightarrow j \rightarrow \cdots \rightarrow j' \rightarrow \ell' \\
&i \rightarrow i' \rightarrow k \rightarrow \cdots \rightarrow k' \rightarrow j \\
&j \rightarrow j' \rightarrow \ell \rightarrow \cdots \rightarrow \ell' \\
&\ell \rightarrow \ell' \\
&c_{kj} \neq c_{ki} \\
&c_{kj'} \neq c_{ki} \\
&c_{ki} = 1 \\
&c_{\ell k} \neq c_{ki} \\
&c_{\ell'k} \neq c_{ki}
\end{align*}
\]
Example

Postroute (-2, 2) ➞ 8 nonunit edges
Example

Postroute (-2, 2)

Normalize (2, 4)

Preroute (2, 3)

Back Eliminate (2, 1)

Front Eliminate (1, 2)

Normalize (-2, 1)

⇒ 8 nonunit edges
Example

Postroute (-2, 2)
Front Eliminate (1, 2)
Example

Postroute (-2, 2)
Front Eliminate (1, 2)
Preroute (2, 3)

⇒ 8 nonunit edges
Example

- Postroute (-2, 2)
- Front Eliminate (1, 2)
- Preroute (2, 3)
- Back Eliminate (2, 1)

⇒ 8 nonunit edges
Example

Postroute (-2, 2)
Front Eliminate (1, 2)
Preroute (2, 3)
Back Eliminate (2, 1)
Normalize (-2, 1)

⇒ 8 nonunit edges
Example

- 3
- 4
- 5
- 1
- 2
- -2
- -1
- 0

- Postroute (-2, 2)
- Front Eliminate (1, 2)
- Preroute (2, 3)
- Back Eliminate (2, 1)
- Normalize (-2, 1)
- Normalize (2, 4)

⇒ 8 nonunit edges
Example

Postroute (-2, 2)
Front Eliminate (1, 2)
Preroute (2, 3)
Back Eliminate (2, 1)
Normalize (-2, 1)
Normalize (2, 4)

⇒ 8 nonunit edges
Are divisions Useful?

Possibly, but we don’t know how to use them

Greedy Algorithm - reroutings lead to *small* improvement at great cost (still not as good as randomized algorithm)

Randomized algorithm - reroutings don’t appear to help
Outline

Introduction and Motivation
  Linearization
  Vector Propagation
  Preaccumulation

Jacobian Scarcity
  Structural Properties of Jacobians
  Randomized Algorithms
  Results
  Rerouting and Normalization

Future Work
  Connections to Optimal Jacobian Accumulation
  Leveraging Face Elimination
Scarsity $\leftrightarrow$ Optimal Jacobian Accumulation

\[ \dot{x}_1 \dot{x}_2 \dot{x}_p - 2 - 1 \]

- Combined cost metric: preaccumulation followed by forward vertex elim.
- Results for OJA apply (modulo the fact that $p$ tends to infinity)
- Implication: divisions are useful for OJA (or not? face elim.?)
Scarsity ↔ Optimal Jacobian Accumulation

- Combined cost metric: preaccumulation followed by forward vertex elim.
- Results for OJA apply (modulo the fact that \( p \) tends to infinity)
- Implication: divisions are useful for OJA (or not? face elim.?)
Exploiting Scarsity with Face Elimination
Exploiting Scarsity with Face Elimination

Eliminating face $(c_{1-1}, c_{21})$
Exploiting Scarsity with Face Elimination

-1 \rightarrow 2 \rightarrow -1

elim. face \( (c_{1-1}, c_{21}) \)

-1 \rightarrow 0 \rightarrow 3

elim. face \( (c_{1-1}, c_{21}) \)